Lecture 16: Language Model

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, and Chris Tanner



Outline

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

We could easily spend an entire semester on this material.

The goal for today and Wednesday is to convey:

- the ubiquity and importance of sequential data
- high-level overview of the most useful, relevant models
- foundation for diving deeper
- when to use which models, based on your data

Outline

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions

Background

Regardless of how we model sequential data, keep in mind that we can estimate any time series as follows:

This compounds for all subsequent events, too

$$P(x_1, \dots, x_T) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \dots, x_1)$$

Joint distribution of all measurements

Conditional probability of an event, depends on all of the events that occurred before it.

Example

The probability of the following 3-day weather pattern in Seattle:

Day 1

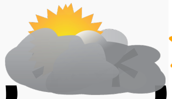
Day 2

Day 3

Example

The probability of the following 3-day weather pattern in Seattle:

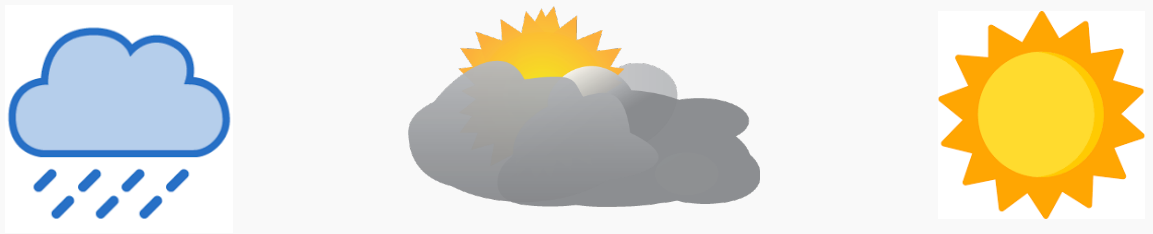
Day 1	Day 2	Day 3
		

$P($  ,  ,  $) =$

Example

The probability of the following 3-day weather pattern in Seattle:

Day 1 Day 2 Day 3



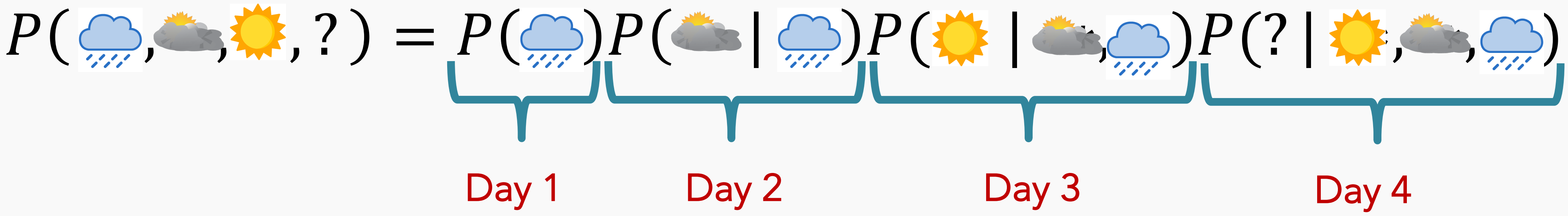
$$P(\text{🌧},\text{⛅},\text{☀}) = \underbrace{P(\text{🌧})}_{\text{Day 1}}\underbrace{P(\text{⛅} \mid \text{🌧})}_{\text{Day 2}}\underbrace{P(\text{☀} \mid \text{⛅},\text{🌧})}_{\text{Day 3}}$$

Example

Why is it useful to accurately estimate the joint of any given sequence of length N?

Background

Having the ability to estimate the probability of any sequence of length N allows us to determine the most likely next event (i.e., sequence of length $N+1$)



For the remainder of this lecture, we will use text (natural language) as examples because:

- It's easy to interpret success/failures
- Real-world impact and commonplace usages
- Availability of data to try things yourself

Yet, for any model, you can imagine using any other sequential data

Language Modelling

A Language Model represents the language used by a given entity (e.g., a particular person, genre, or other well-defined class of text)

Language Modelling

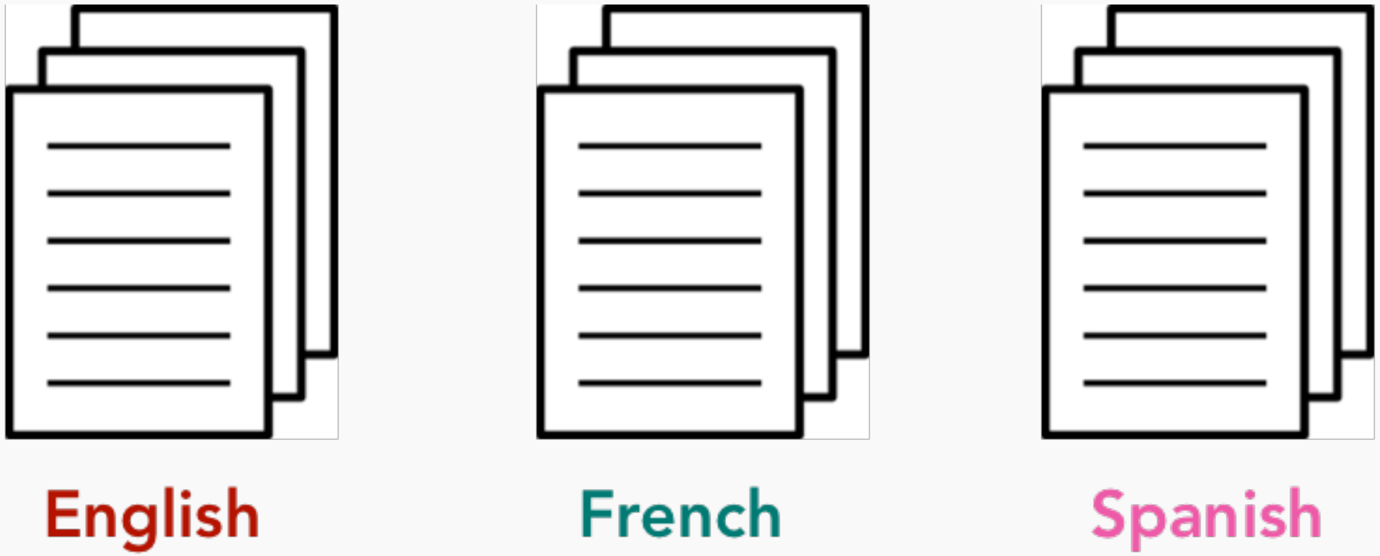
A Language Model represents the language used by a given entity (e.g., a particular person, genre, or other well-defined class of text)

Spam

Not Spam

Language Modelling

A Language Model represents the language used by a given entity (e.g., a particular person, genre, or other well-defined class of text)



Language Modelling: **Formal Definition**

A Language Model estimates the probability of any sequence of words

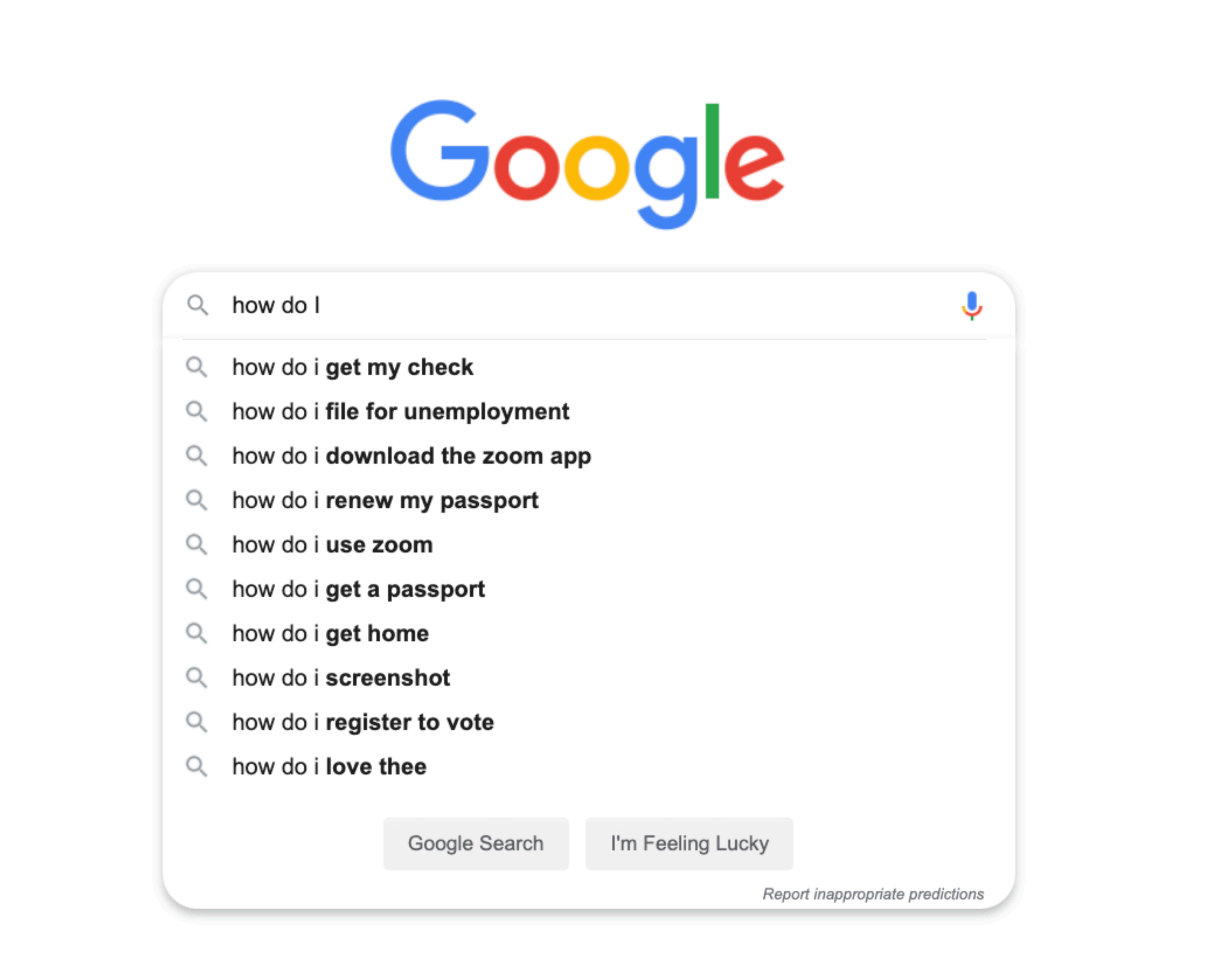
Let $\boldsymbol{X}$ = "Eleni was late for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

$\mathrm{P}(\boldsymbol{X}) = P($"Eleni was late for class"$)$

Language Modelling

Generate Text

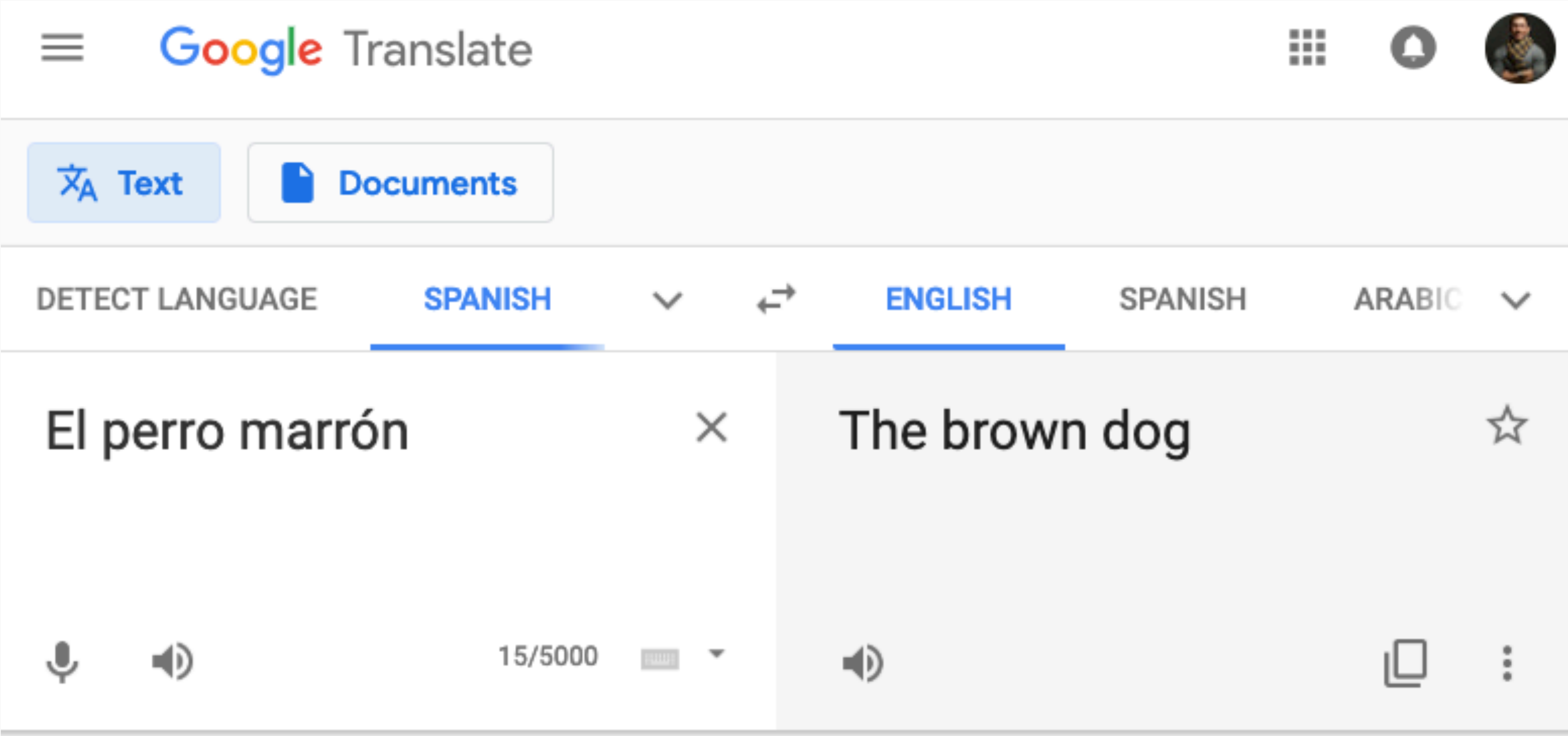


Language Modelling

Generate Text

Language Modelling

Generate Text



Language Modelling

"Drug kingpin El Chapo testified that he gave MILLIONS to Pelosi, Schiff & Killary. The Feds then closed the courtroom doors."

Language Modelling

A Language Model is useful for:

Generating Text

- Auto-complete
- Speech-to-text
- Question-answering / chatbots
- Machine translation

Classifying Text

- Authorship attribution
- Detecting spam vs not spam

And much more!

Language Modelling

Today, we heavily focus on Language Modelling (LM) because:

1. It's foundational for nearly all NLP tasks

2. Since we're ultimately modelling a sequence, LM approaches are generalizable to any type of data, not just text.

Language Modelling: **unigrams**

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others.

Count how often each word occurs (in the training data).

Language Modelling: **unigrams**

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others

Let $\boldsymbol{X}$ = "Eleni was late for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

Language Modelling: **unigrams**

How can we build a language model?

Naive Approach: unigram model

Assume each word is independent of all others

Let $\boldsymbol{X}$ = "Eleni was late for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

You calculate each of these probabilities from the training corpus

$$\mathrm{P}(\boldsymbol{X}) = P(\text{Eleni})P(\text{was})P(\text{late})P(\text{for})P(\text{class})$$

$= 0.00015 * 0.01 * 0.004 * 0.03 * 0.0035$
$= 6.3\text{x}10^{-13}$

Language Modelling: **unigrams**

UNIGRAM ISSUES?

Language Modelling: **unigrams**

UNIGRAM ISSUES?

Context doesn't play a role at all

$$P(\text{"Eleni was late for class"}) = P(\text{"class for was late Eleni"})$$

Language Modelling: **unigrams**

UNIGRAM ISSUES?

Context doesn't play a role at all

$$P(\text{"Eleni was late for class"}) = P(\text{"class for was late Eleni"})$$

Sequence generation: What's the most likely next word?

Eleni was late for class _____

Language Modelling: **unigrams**

UNIGRAM ISSUES?

Context doesn't play a role at all

$$P(\text{"Eleni was late for class"}) = P(\text{"class for was late Eleni"})$$

Sequence generation: What's the most likely next word?

Eleni was late for class _____

Eleni was late for class <u>the</u>

Anqi was late for class the <u>the</u>

Language Modelling: **bigrams**

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

Let $\boldsymbol{X}$ = "Eleni was late for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

Language Modelling: **bigrams**

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Eleni was late for class"

w_1 w_2 w_3 w_4 w_5

$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Eleni})$

Language Modelling: **bigrams**

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Eleni $\boxed{\text{was late}}$ for class"

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5$

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Eleni})P(\text{late}|\text{was})$$

Language Modelling: **bigrams**

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Eleni was late for class"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Eleni})P(\text{late}|\text{was})P(\text{for}|\text{late})$$

Language Modelling: **bigrams**

How can we build a language model?

Alternative Approach: bigram model

Look at *pairs* of consecutive words

probability

Let $\boldsymbol{X}$ = "Eleni was late $\boxed{\text{for class}}$"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Eleni})P(\text{late}|\text{was})P(\text{for}|\text{late})P(\text{class}|\text{for})$$

Language Modelling: **bigrams**

How can we bu

You calculate each of these probabilities by simply counting the occurrences

$$P(class \mid for) = \frac{count(\textbf{for class})}{count(\textbf{for})}$$

probability

Let $\boldsymbol{X}$ = "Eleni was late $\boxed{\text{for class}}$"

w_1 w_2 w_3 w_4 w_5

$$\mathrm{P}(\boldsymbol{X}) = P(\text{was}|\text{Eleni})P(\text{late}|\text{was})P(\text{for}|\text{late})P(\text{class}|\text{for})$$

Language Modelling: **bigrams**

BIGRAM ISSUES?

Language Modelling: **bigrams**

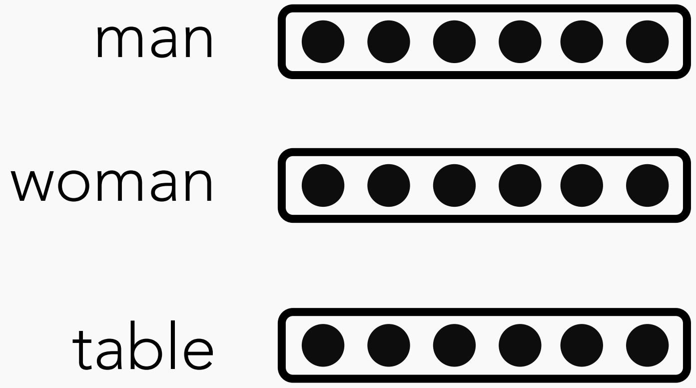
BIGRAM ISSUES?

- **Out-of-vocabulary** items are 0 → kills the overall probability
- Always need more context (e.g., trigram, 4-gram), but **sparsity** is an issue (rarely seen subsequences)
- **Storage** becomes a problem as we increase window size
- No semantic information conveyed by counts (e.g., vehicle vs car)

Language Modelling: **neural networks**

IDEA: Let's use a neural networks!

First, each word is represented by a word embedding (e.g., vector of length 200)



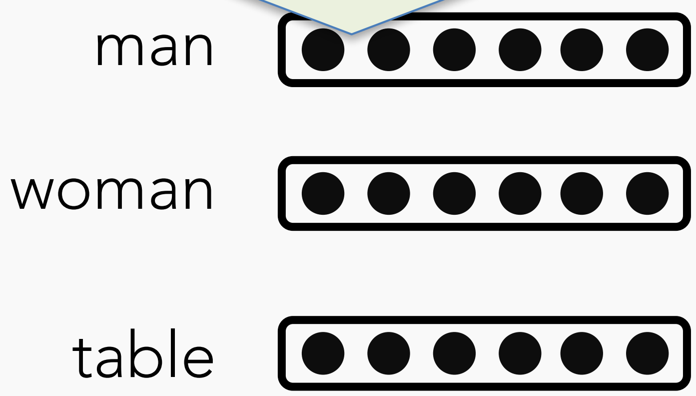
Language Modelling: **neural networks**

IDEA: Let's use

First, each wo

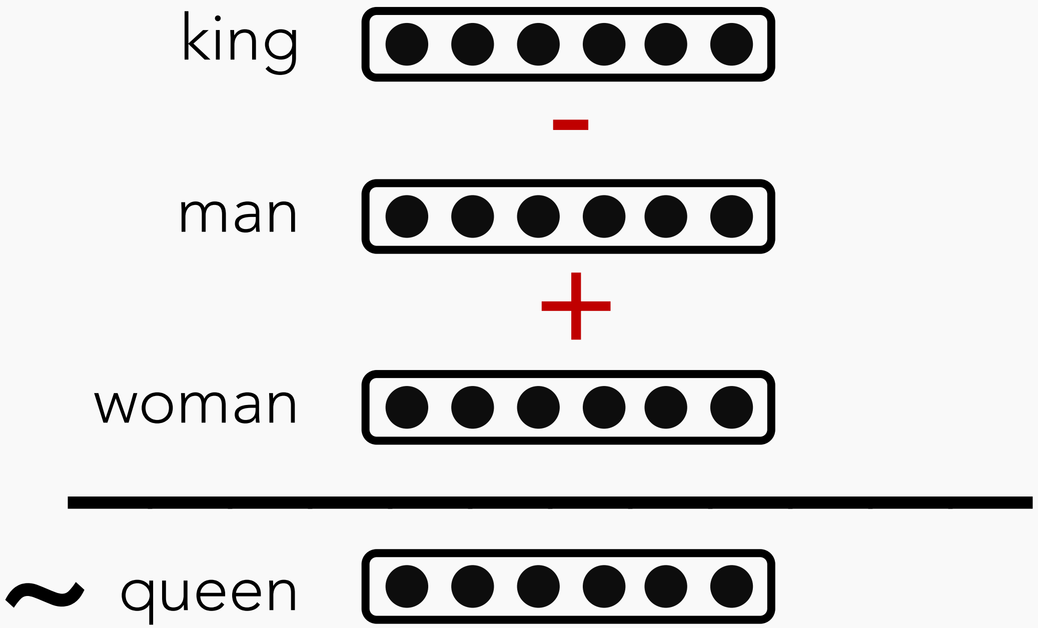
(e.g., vector of

- Each circle is a specific *floating point* scalar
- Words that are more *semantically similar* to one another will have **embeddings** that are proportionally similar, too
- We can *use pre-existing* word embeddings that have been trained on gigantic corpora



Language Modelling: **neural networks**

These word embeddings are so rich that you get nice properties:



Word2vec: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
GloVe: https://www.aclweb.org/anthology/D14-1162.pdf

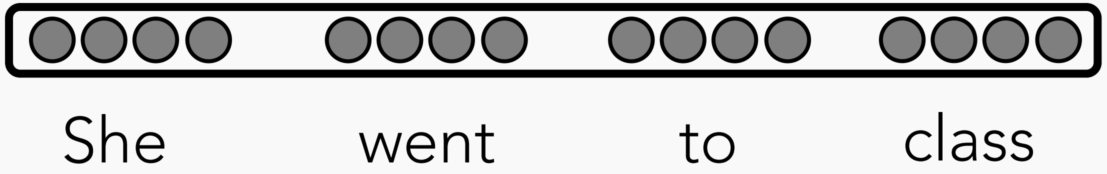
Language Modelling: **neural networks**

How can we use these embeddings to build a LM?

Remember, we only need a system that can estimate:

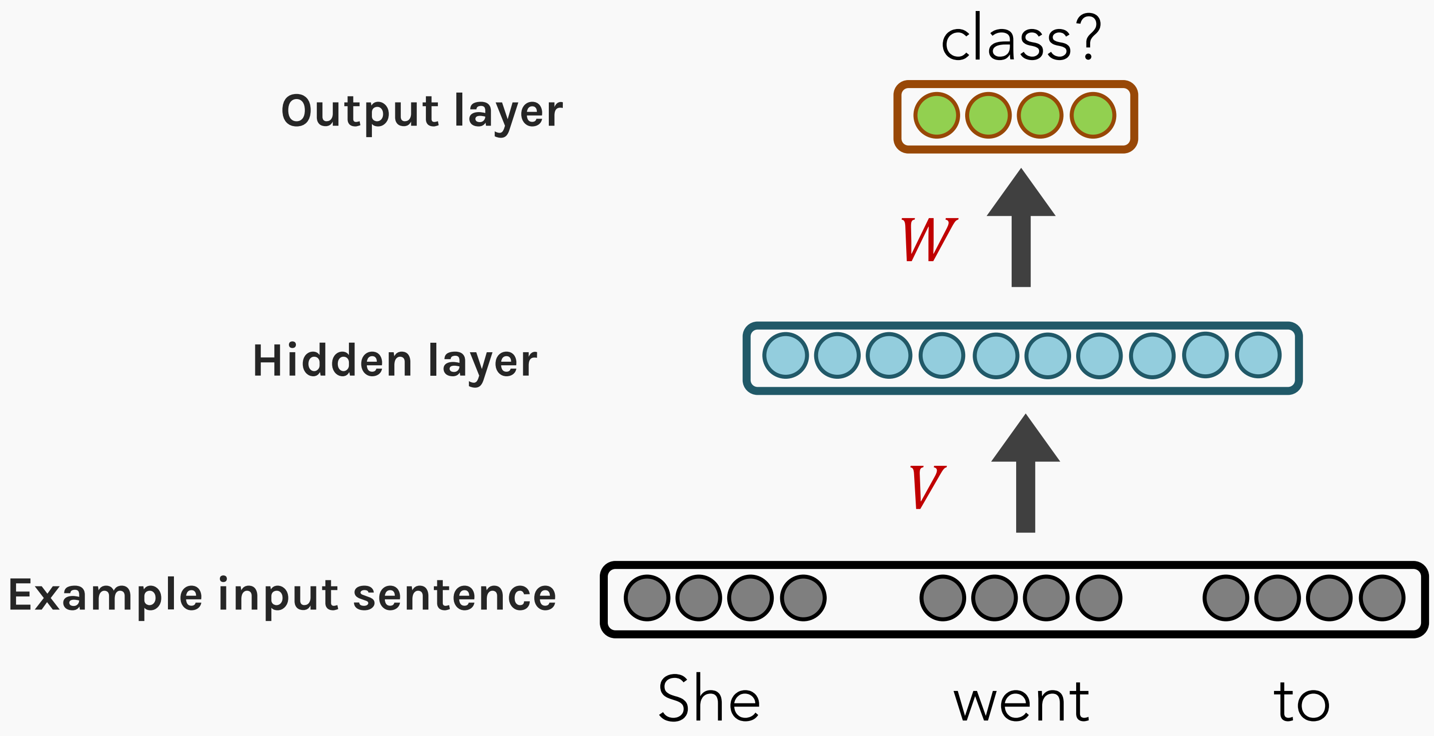
$$P(\underbrace{x_{t+1}}_{\text{next word}}|\underbrace{x_t, x_{t-1}, \dots, x_1}_{\text{previous words}})$$

Example input sentence



Neural Approach #1: Feed-forward Neural Net

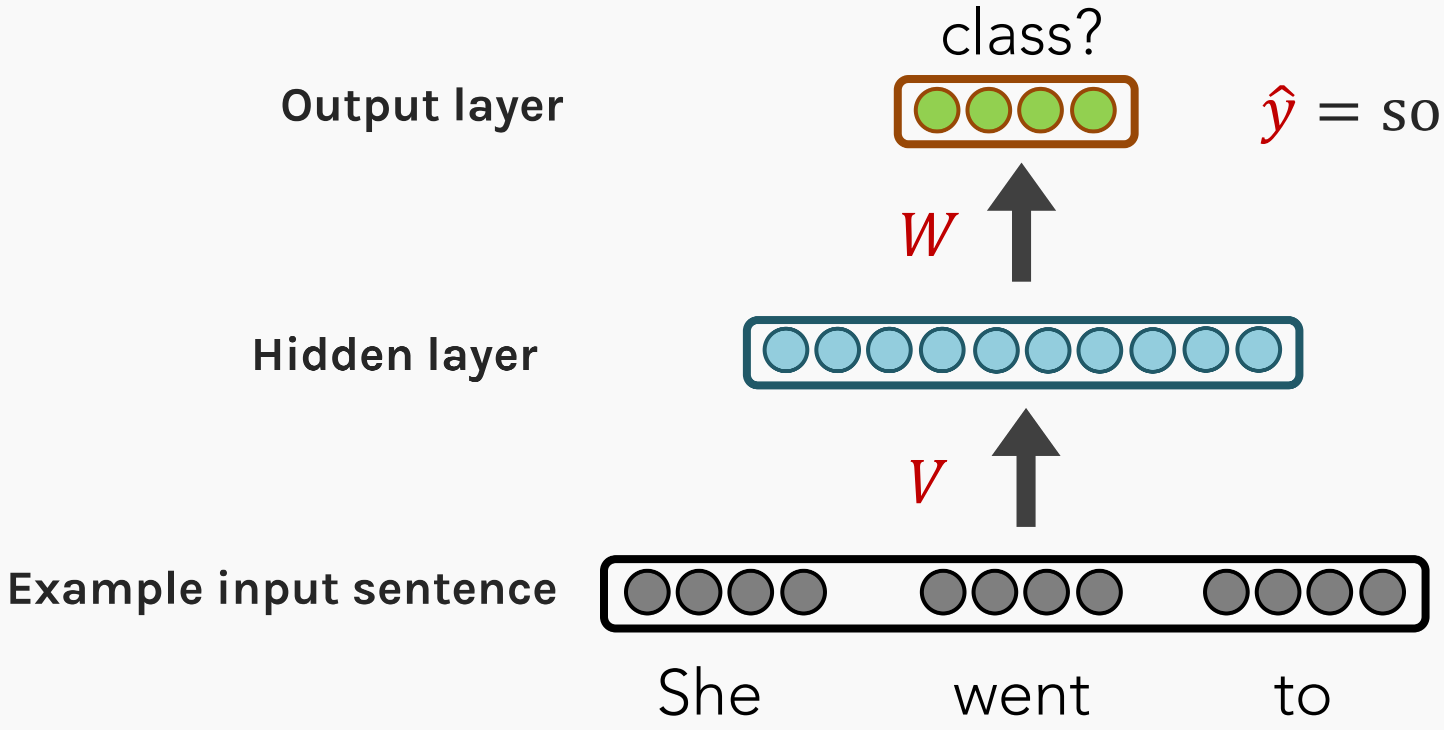
General Idea: using *windows* of words, predict the next word



Language Modelling: **Feed-forward Neural Net**

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word



$$\hat{y} = \mathrm{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$$

$$h = f(Vx + b_1)$$

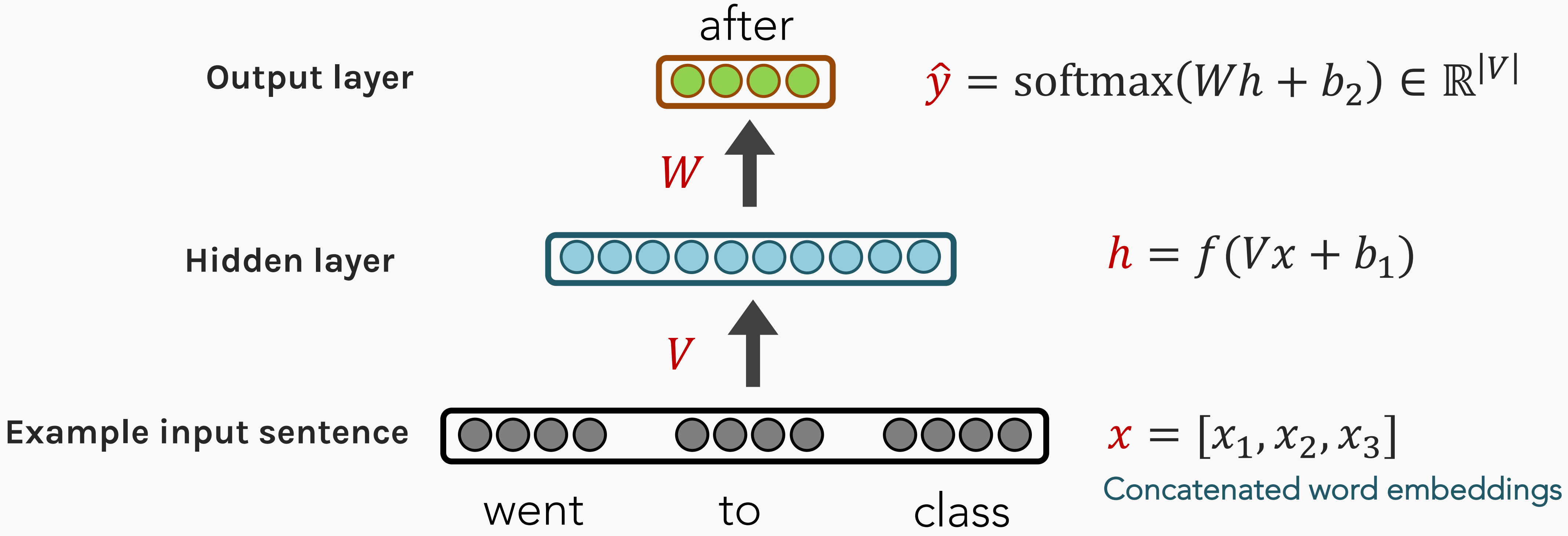
$$x = [x_1, x_2, x_3]$$

Concatenated word embeddings

Language Modelling: **Feed-forward Neural Net**

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word



Language Modelling: **Feed-forward Neural Net**

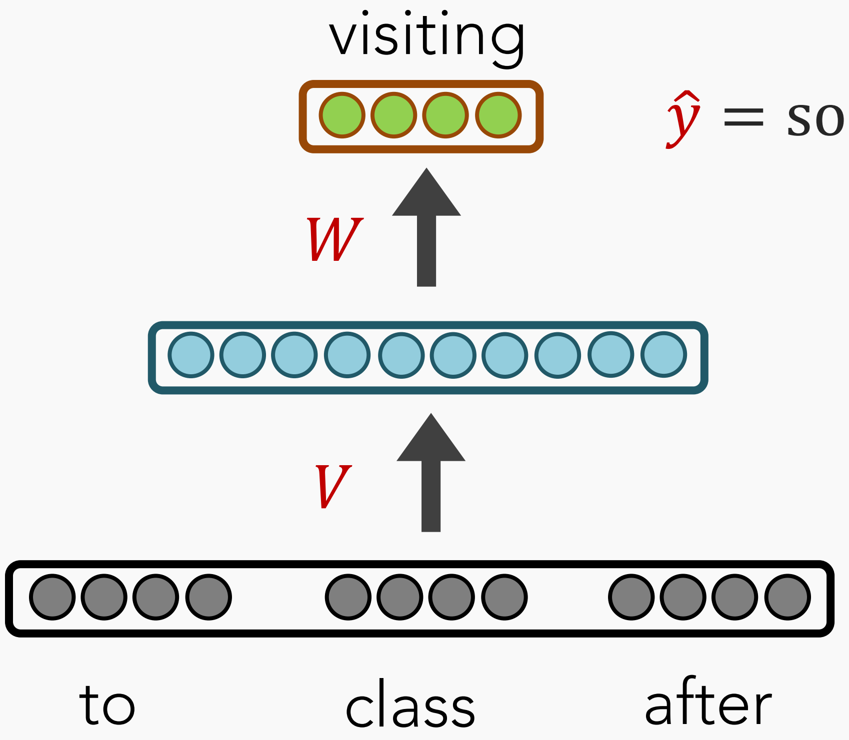
Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word

Output layer

Hidden layer

Example input sentence



$\hat{y} = \text{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$

$h = f(Vx + b_1)$

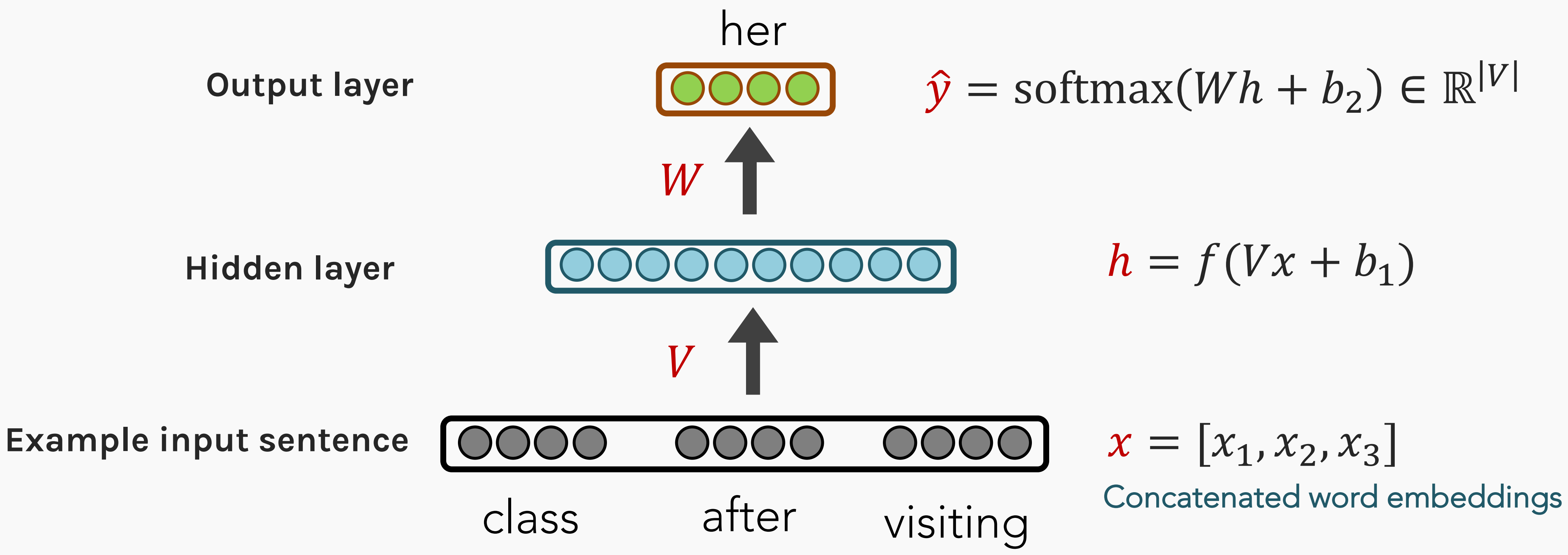
$x = [x_1, x_2, x_3]$

Concatenated word embeddings

Language Modelling: **Feed-forward Neural Net**

Neural Approach #1: Feed-forward Neural Net

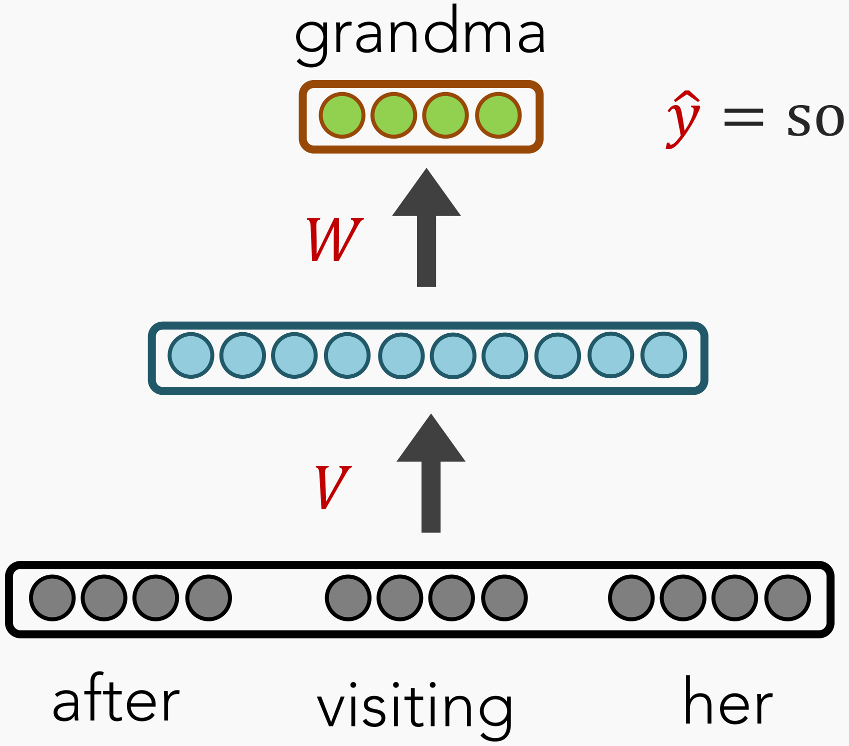
General Idea: using *windows* of words, predict the next word



Language Modelling: **Feed-forward Neural Net**

Neural Approach #1: Feed-forward Neural Net

General Idea: using *windows* of words, predict the next word



Output layer $\hat{y} = \mathrm{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$

Hidden layer $h = f(Vx + b_1)$

Example input sentence $x = [x_1, x_2, x_3]$

Concatenated word embeddings

Language Modelling : **Feed-forward Neural Net**

FFNN STRENGTHS?

FFNN ISSUES?

Language Modelling : **Feed-forward Neural Net**

FFNN STRENGTHS?

- No sparsity issues (it's okay if we've never seen a segment of words)
- No storage issues (we never store counts)

FFNN ISSUES?

- Fixed-window size can never be big enough. Need more context.
- Increasing window size adds many more weights
- The weights awkwardly handle word position
- No concept of time
- Requires inputting entire context just to predict one word

Language Modelling

We especially need a system that:

- Has an “infinite” concept of the past, not just a fixed window
- For each new input, output the most likely next event (e.g., word)

Outline

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

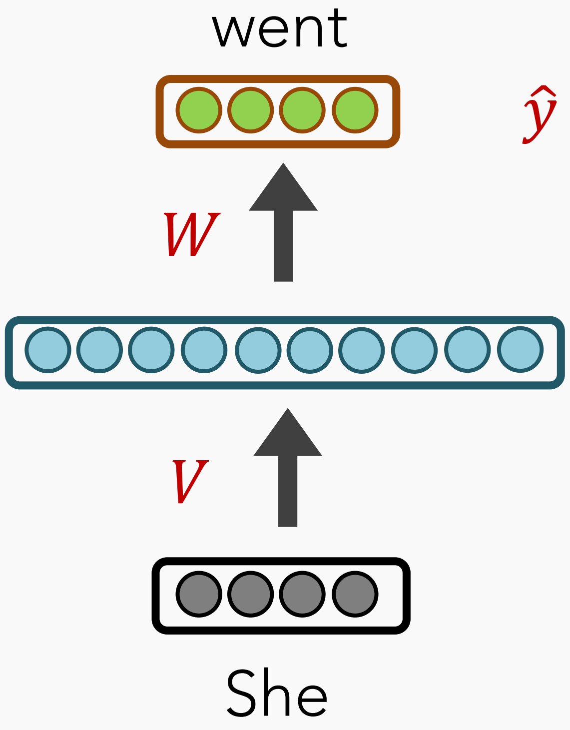
Conclusions


forget gate
previous
cell state
LET ME IN
[adult swim]
LET ME INNNNNN!!!

Language Modelling

IDEA: for every individual input, output a prediction

Let's use the previous hidden state, too



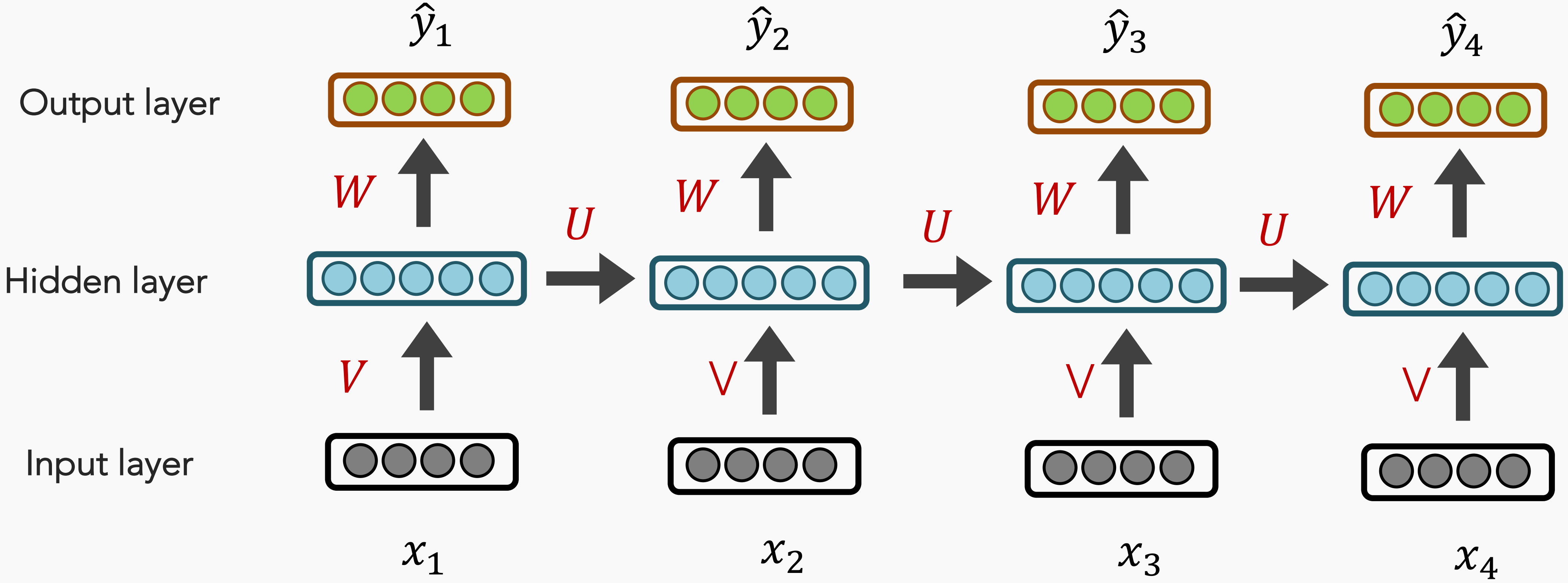
Output layer $\hat{y} = \text{softmax}(Wh + b_2) \in \mathbb{R}^{|V|}$

Hidden layer $h = f(Vx + b_1)$

Example input word $x = x_1$

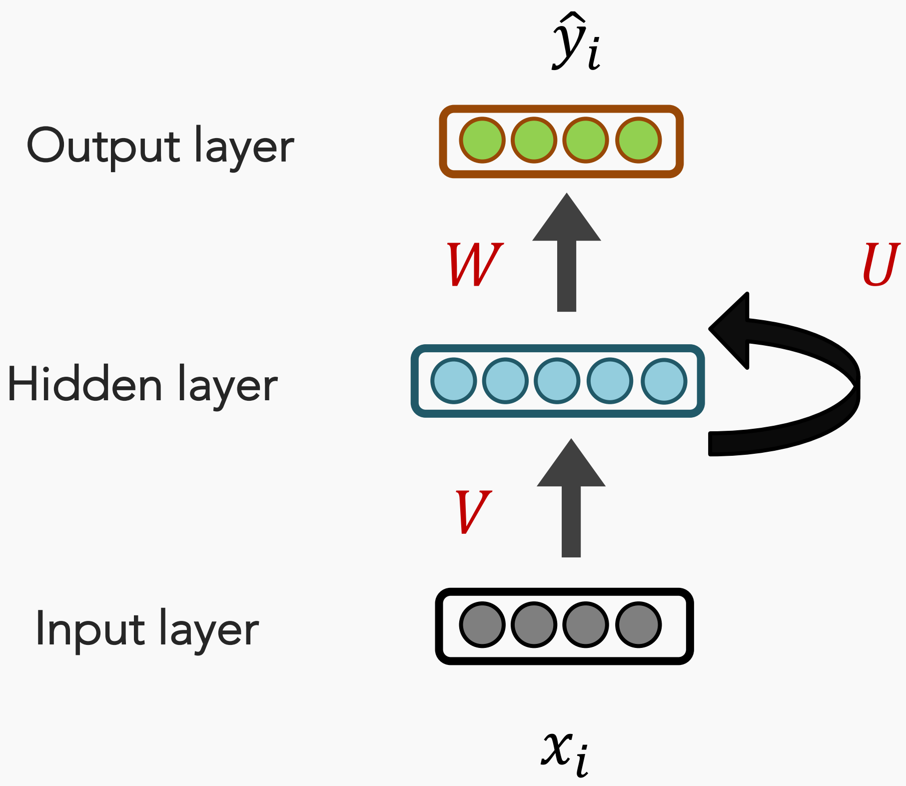
single word embedding

Neural Approach #2: Recurrent Neural Network (RNN)



Language Modelling: **RNNs**

We have seen this abstract view in Lecture 15.

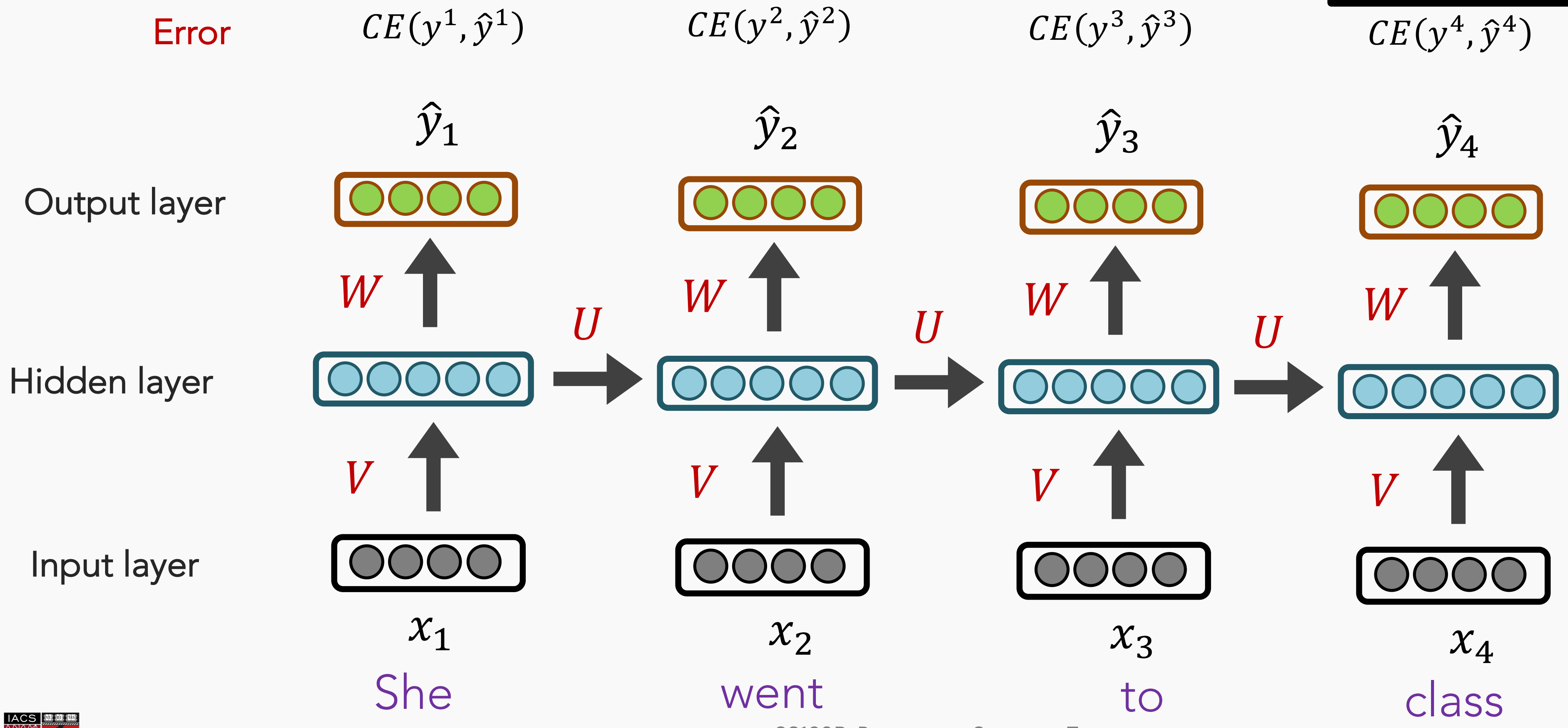


The recurrent loop $\boldsymbol{U}$ conveys that the current hidden layer is influenced by the hidden layer from the previous time step.

RNN (review)

Training Process

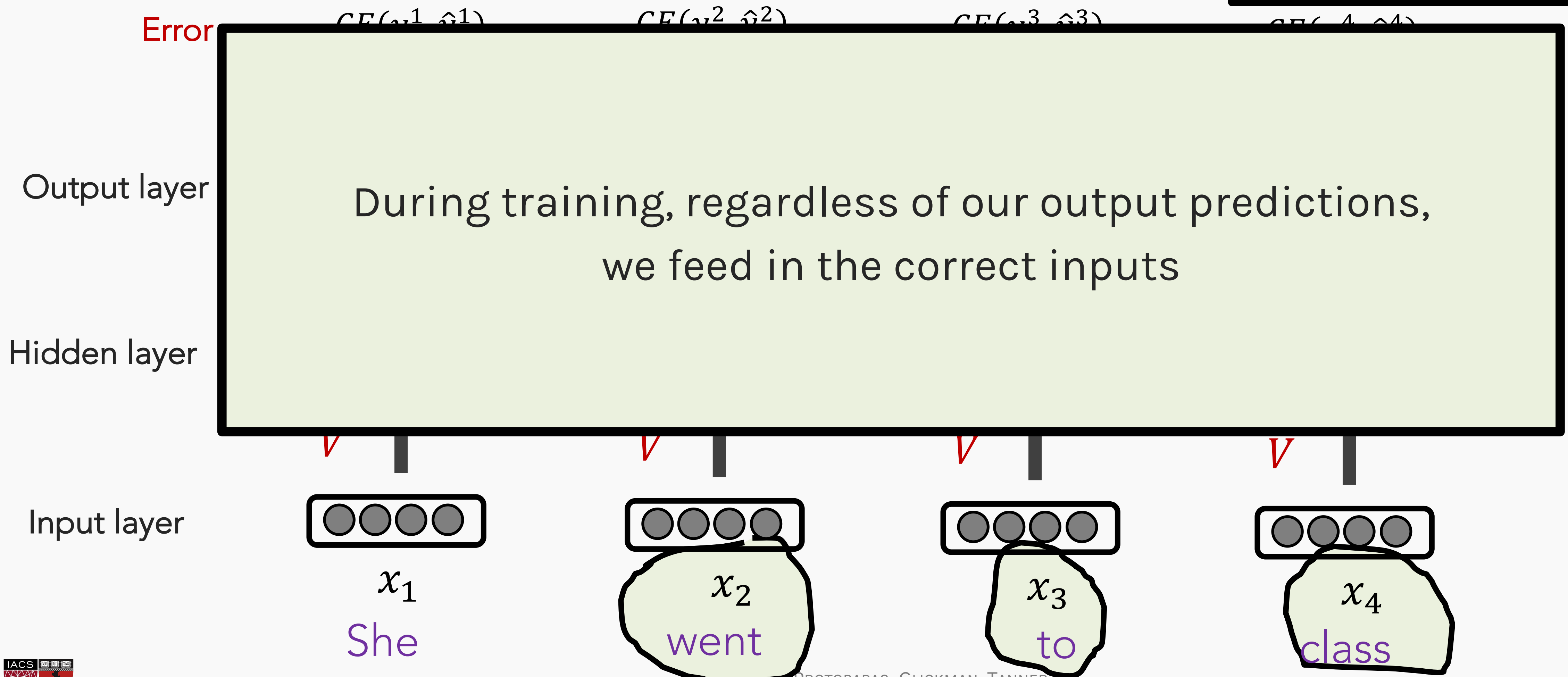
$$CE(y^i, \hat{y}^i) = -\sum_{w \in W} y_w^i \log(\hat{y}_w^i)$$



RNN (review)

Training Process

$$CE(y^i, \hat{y}^i) = -\sum_{w \in W} y_w^i \log(\hat{y}_w^i)$$

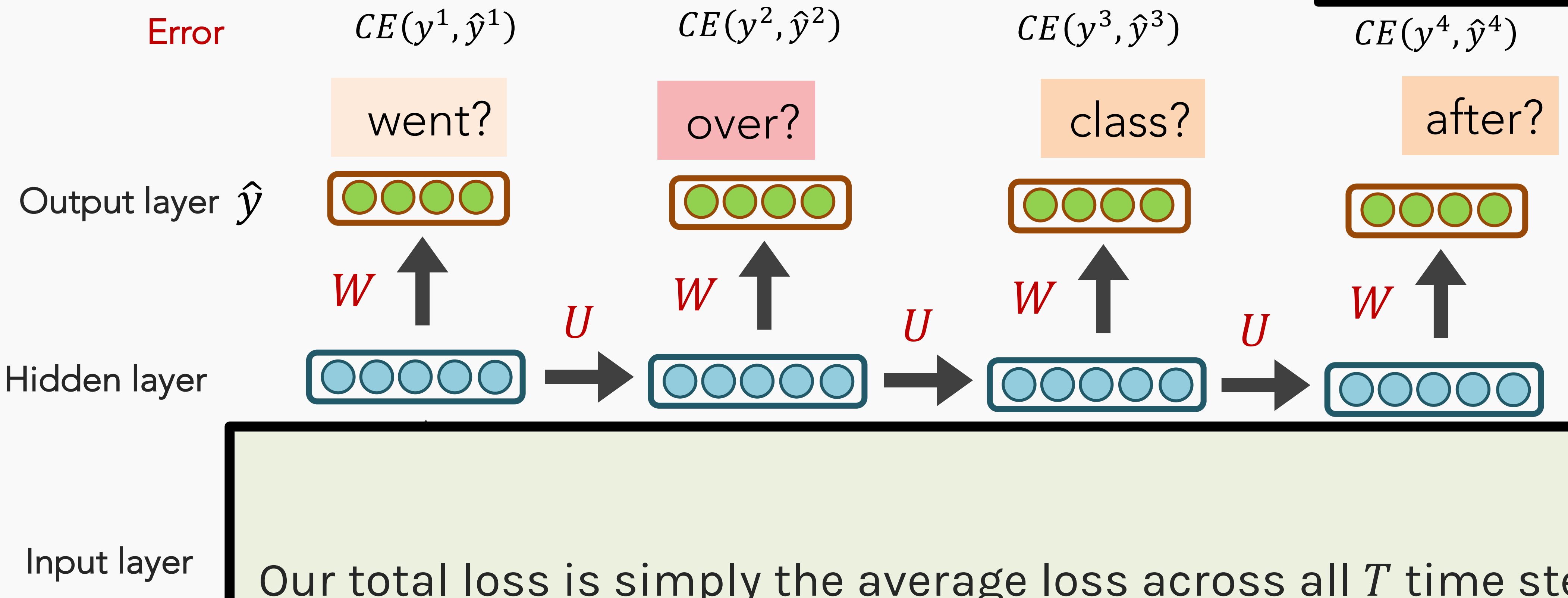


During training, regardless of our output predictions, we feed in the correct inputs

RNN (review)

Training Process

$$CE(y^i, \hat{y}^i) = -\sum_{w \in W} y_w^i \log(\hat{y}_w^i)$$

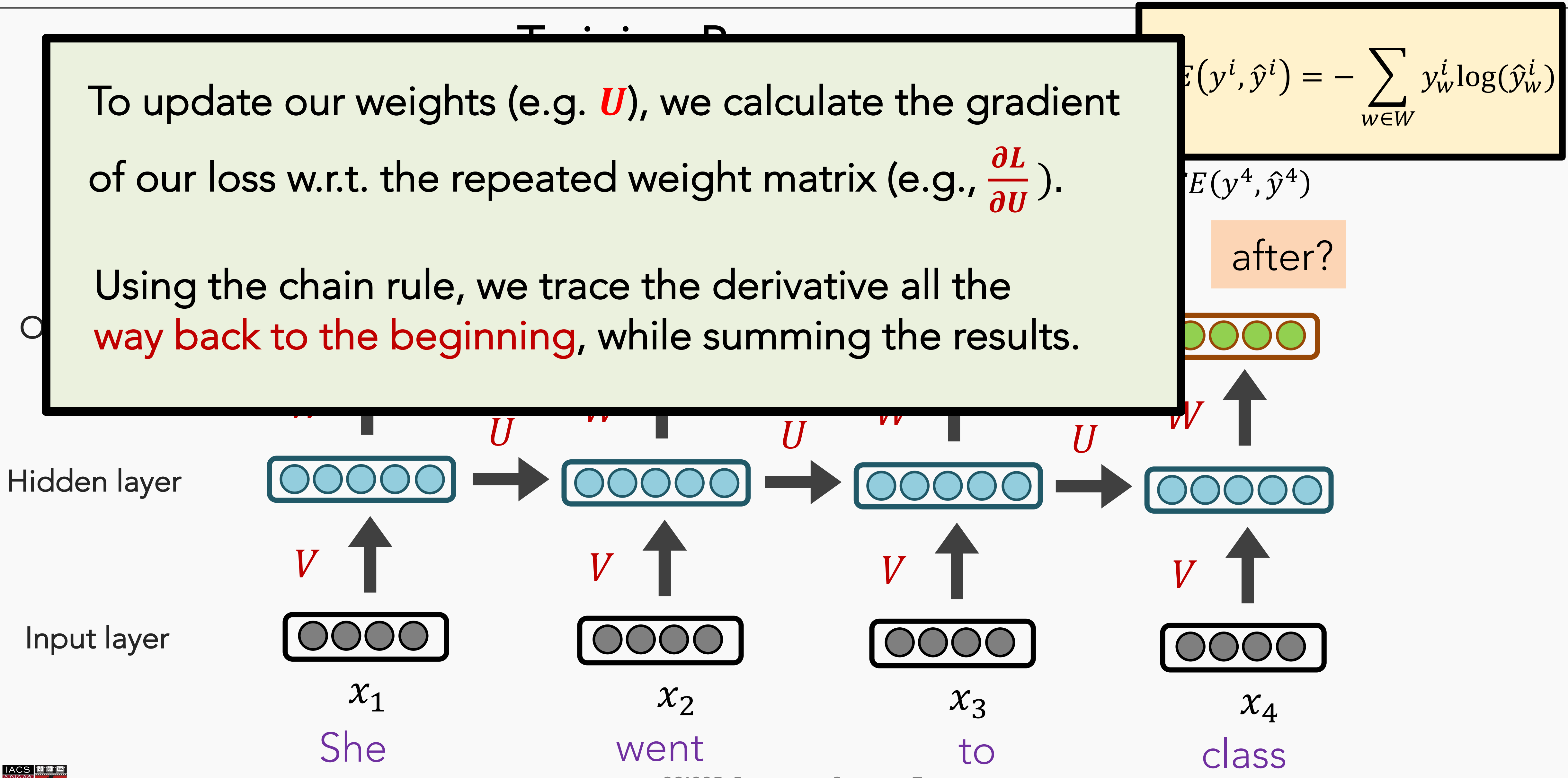


Our total loss is simply the average loss across all T time steps

RNN (review)

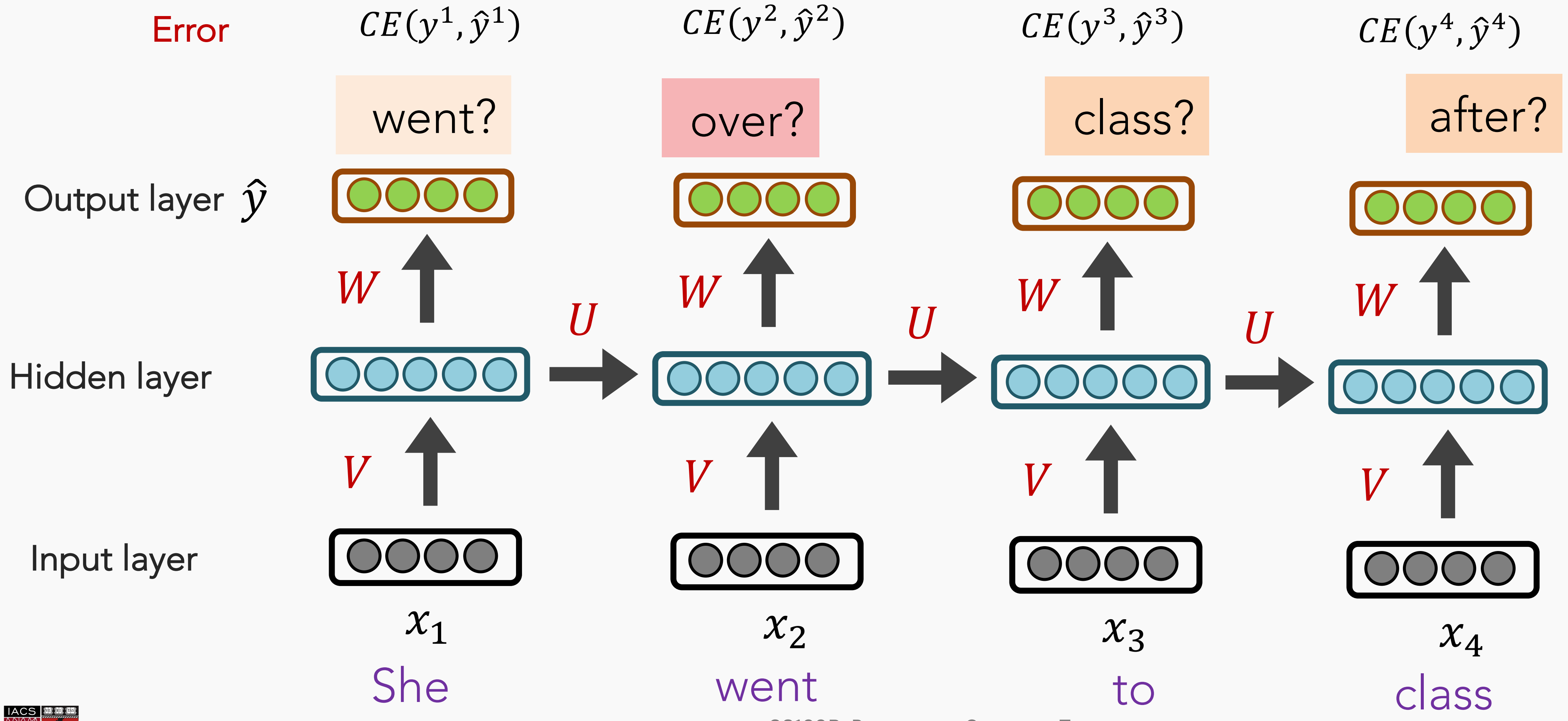
To update our weights (e.g. U), we calculate the gradient of our loss w.r.t. the repeated weight matrix (e.g., $\frac{\partial L}{\partial U}$).

Using the chain rule, we trace the derivative all the way back to the beginning, while summing the results.

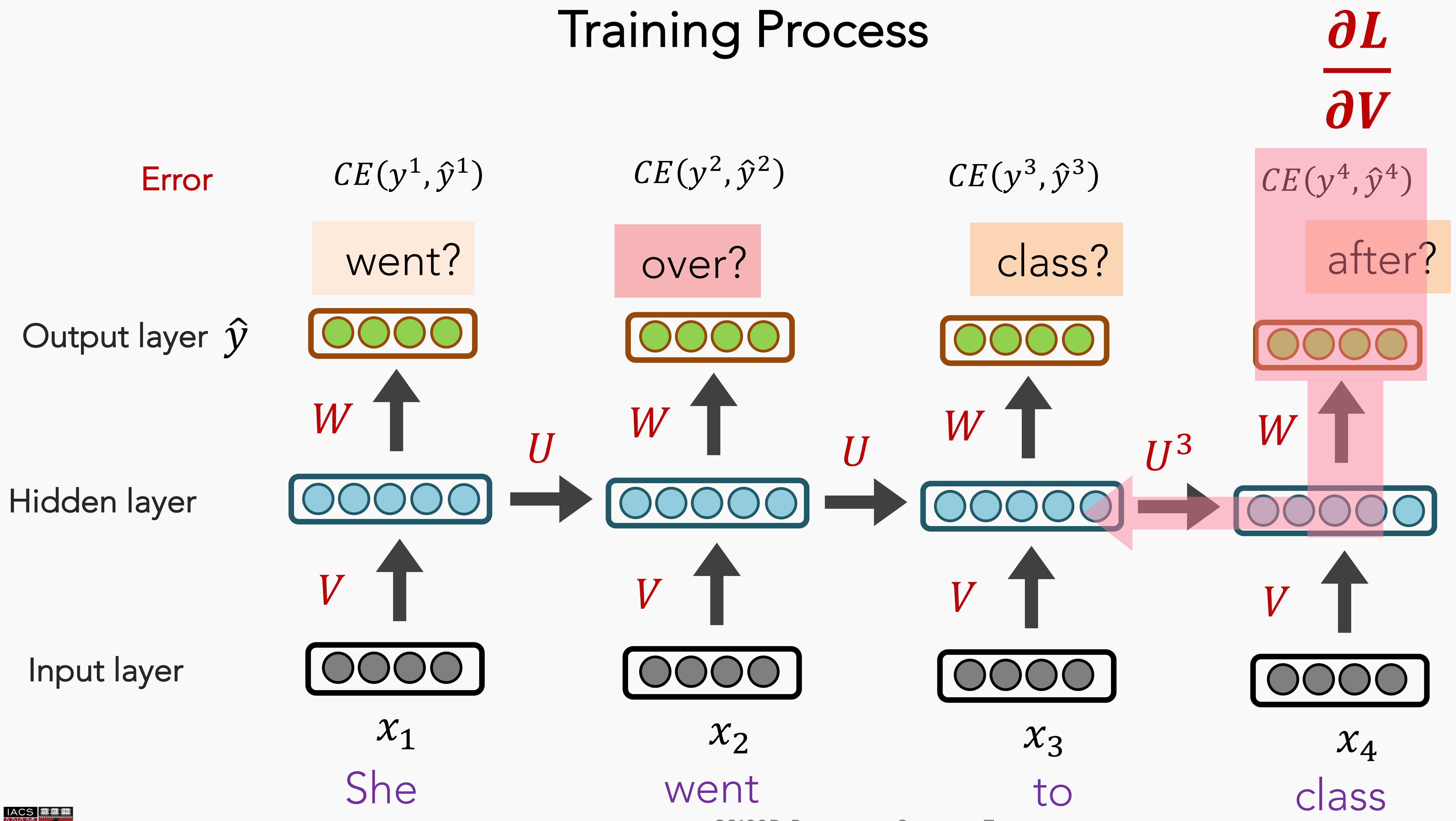


RNN (review)

Training Process

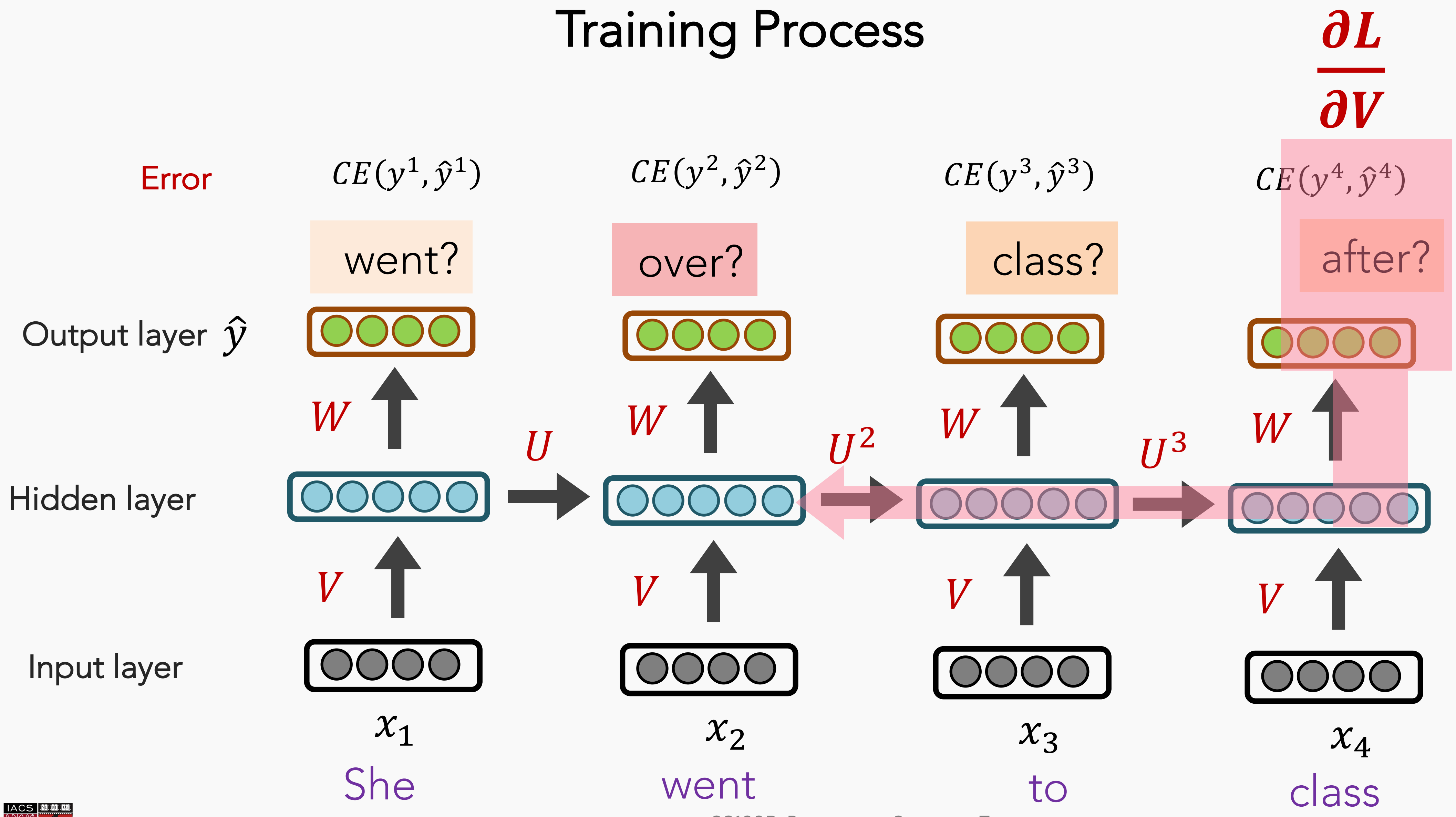


RNN (review)



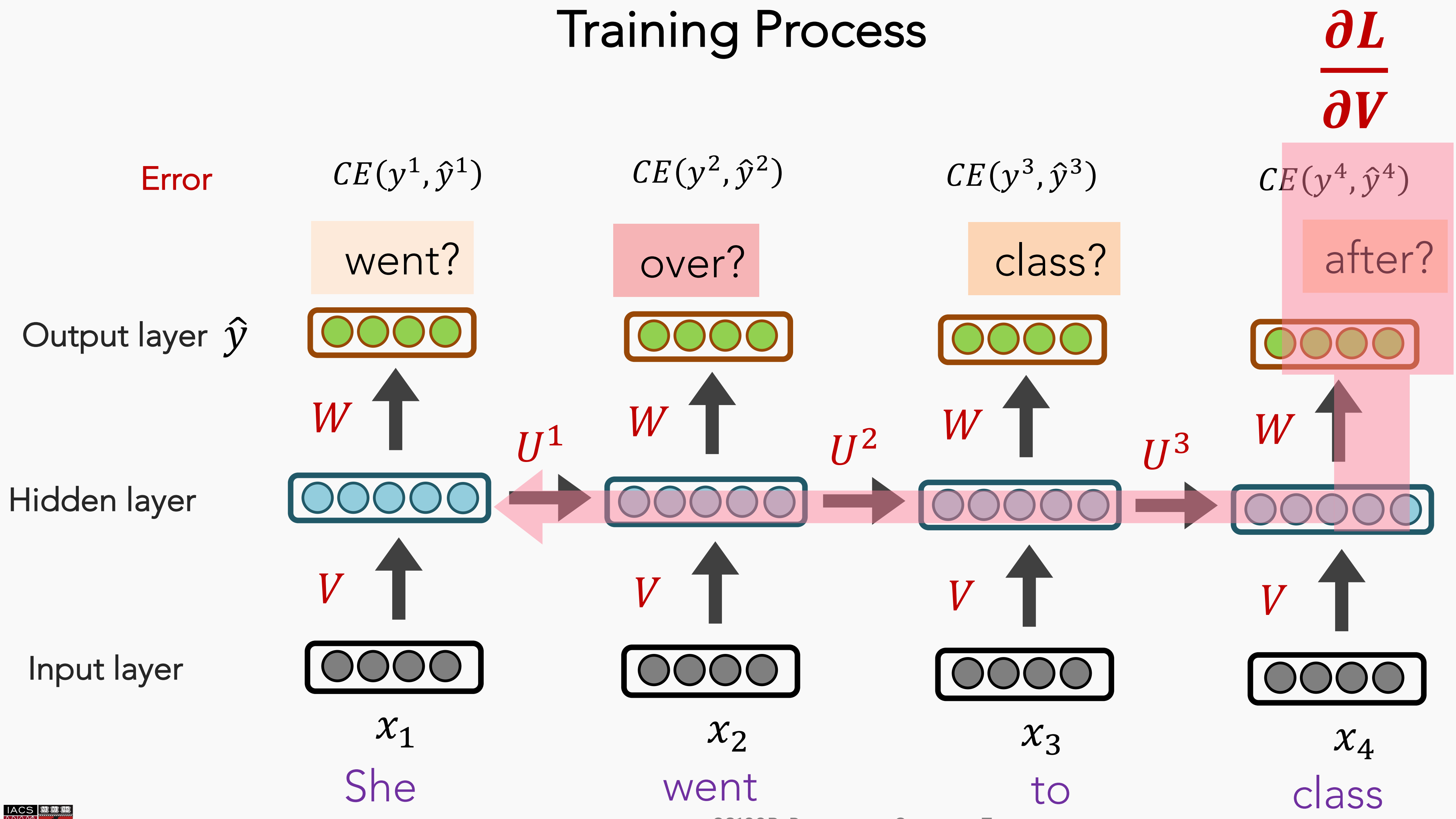
Training Process
$\frac{\partial L}{\partial V}$
Error
$CE(y^1, \hat{y}^1)$
$CE(y^2, \hat{y}^2)$
$CE(y^3, \hat{y}^3)$
$CE(y^4, \hat{y}^4)$
went?
over?
class?
after?
Output layer $\hat{y}$
W
U
U^3
Hidden layer
V
Input layer
x_1
x_2
x_3
x_4
She
went
to
class

RNN (review)



Training Process
$\frac{\partial L}{\partial V}$
Error
$CE(y^1, \hat{y}^1)$
$CE(y^2, \hat{y}^2)$
$CE(y^3, \hat{y}^3)$
$CE(y^4, \hat{y}^4)$
went?
over?
class?
after?
Output layer $\hat{y}$
W
U
U^2
U^3
Hidden layer
V
Input layer
x_1
x_2
x_3
x_4
She
went
to
class

RNN (review)


Training Process
$\frac{\partial L}{\partial V}$
Error
$CE(y^1, \hat{y}^1)$
$CE(y^2, \hat{y}^2)$
$CE(y^3, \hat{y}^3)$
$CE(y^4, \hat{y}^4)$
went?
over?
class?
after?
Output layer $\hat{y}$
W
U^1
U^2
U^3
Hidden layer
V
Input layer
x_1
x_2
x_3
x_4
She
went
to
class

RNN (review)

- This backpropagation through time (BPTT) process is **expensive**
- Instead of updating after every timestep, we tend to do so every T steps (e.g., every sentence or paragraph)
- This isn't equivalent to using only a window size T (a la n-grams) because we still have 'infinite memory'

RNN: Generation

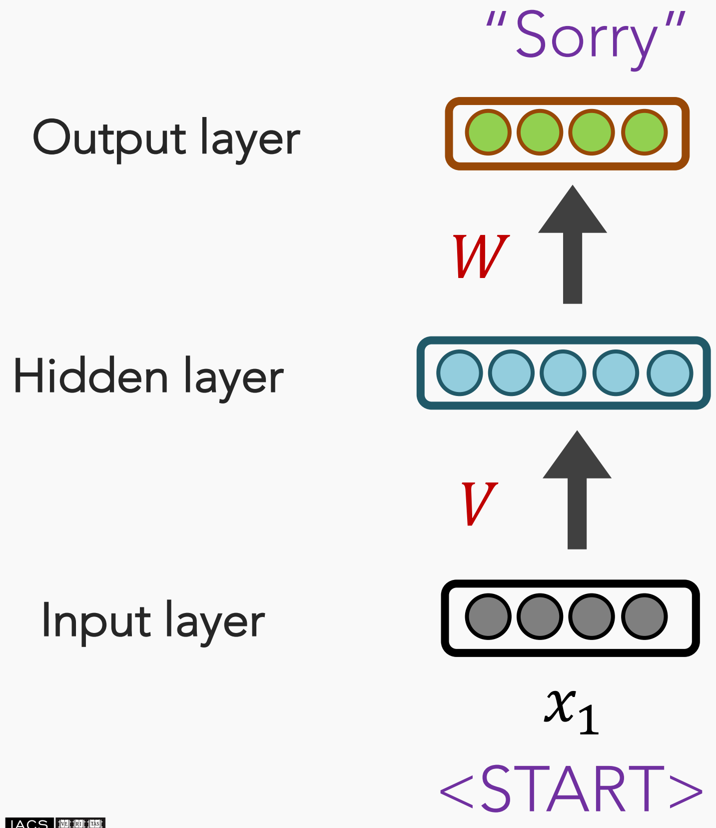
We can generate the most likely next event (e.g., word) by sampling from $\hat{y}$

Continue until we generate <EOS> symbol.

RNN: Generation

We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

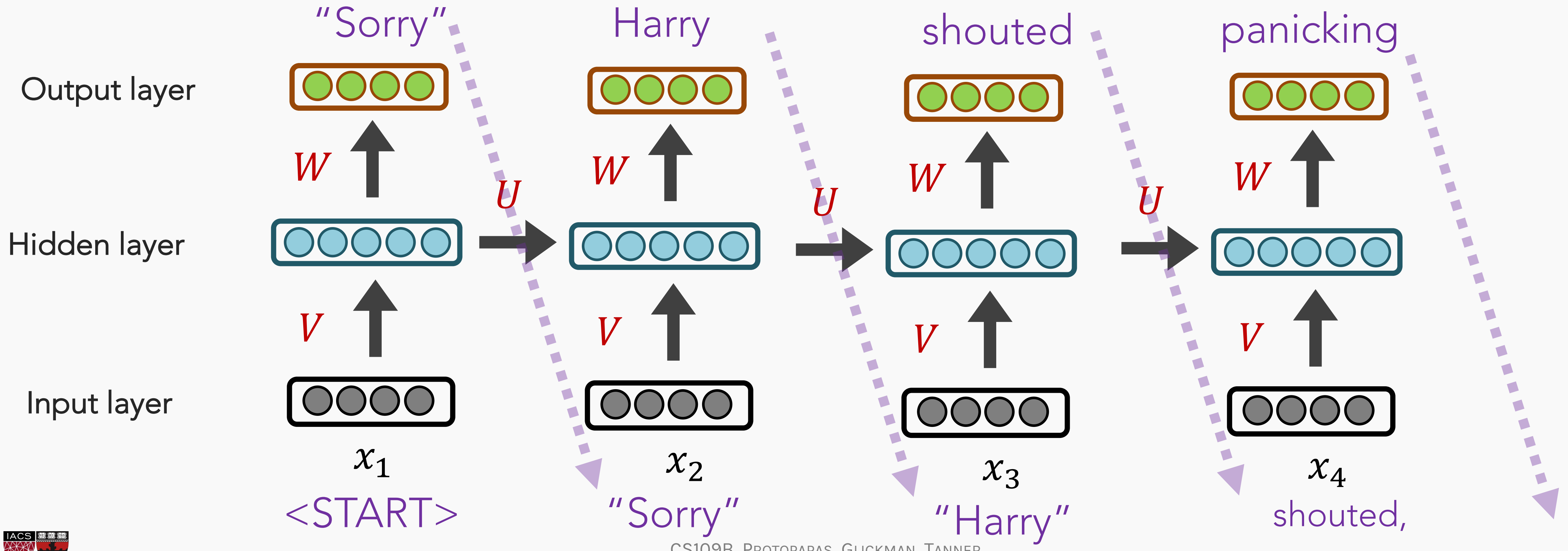
Continue until we generate **<EOS>** symbol.



RNN: Generation

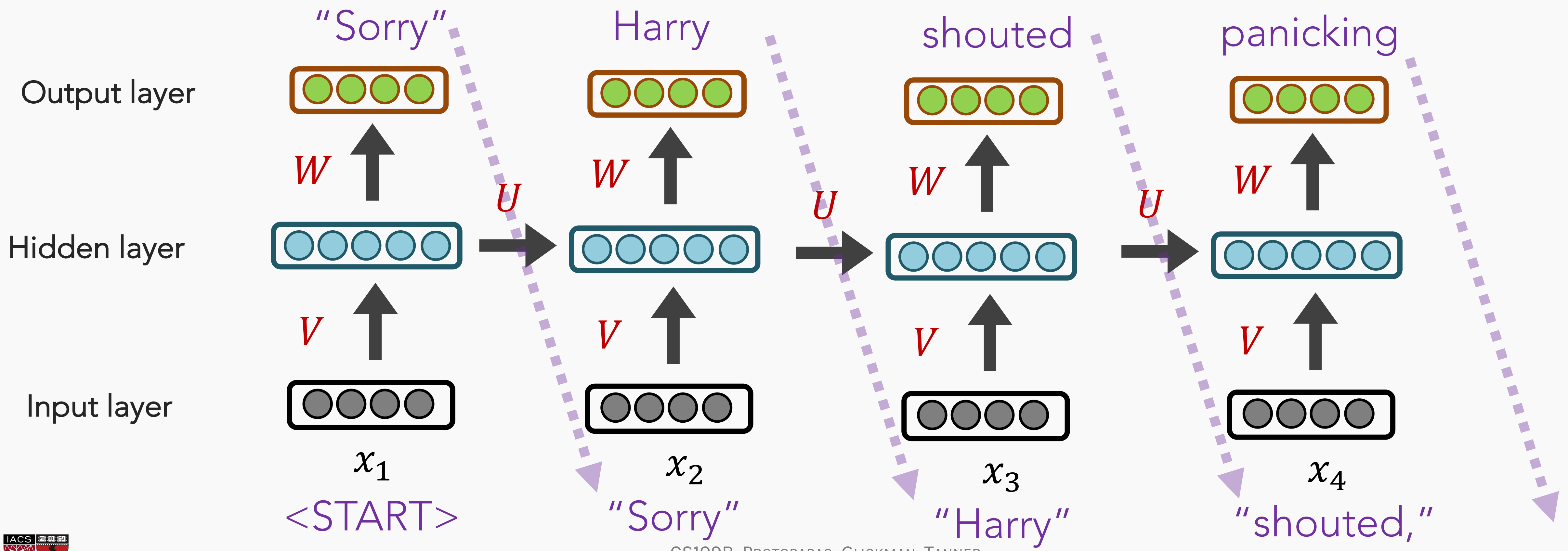
We can generate the most likely **next** event (e.g., word) by sampling from $\hat{y}$

Continue until we generate **<EOS>** symbol.



RNN: Generation

NOTE: the same input (e.g., **"Harry"**) can easily yield different outputs, depending on the context (unlike FFNNs and n-grams).



RNN: Generation

When trained on Harry Potter text, it generates:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

RNN: Generation

When trained on recipes

```
     Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
     Yield: 6 Servings

     2 tb Parmesan cheese -- chopped
     1 c  Coconut milk
     3    Eggs, beaten
```


```
Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients
and stir in the chocolate and pepper.
```

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

RNNs: Overview

RNN STRENGTHS?

- Can handle infinite-length sequences (not just a fixed-window)
- Has a "memory" of the context (thanks to the hidden layer's recurrent loop)
- Same weights used for all inputs, so word order isn't wonky (like FFNN)

RNN ISSUES?

- Slow to train (BPTT)
- Due to "infinite sequence", gradients can easily **vanish** or **explode**
- Has trouble actually making use of long-range context

RNNs: Overview

RNN STRENGTHS?

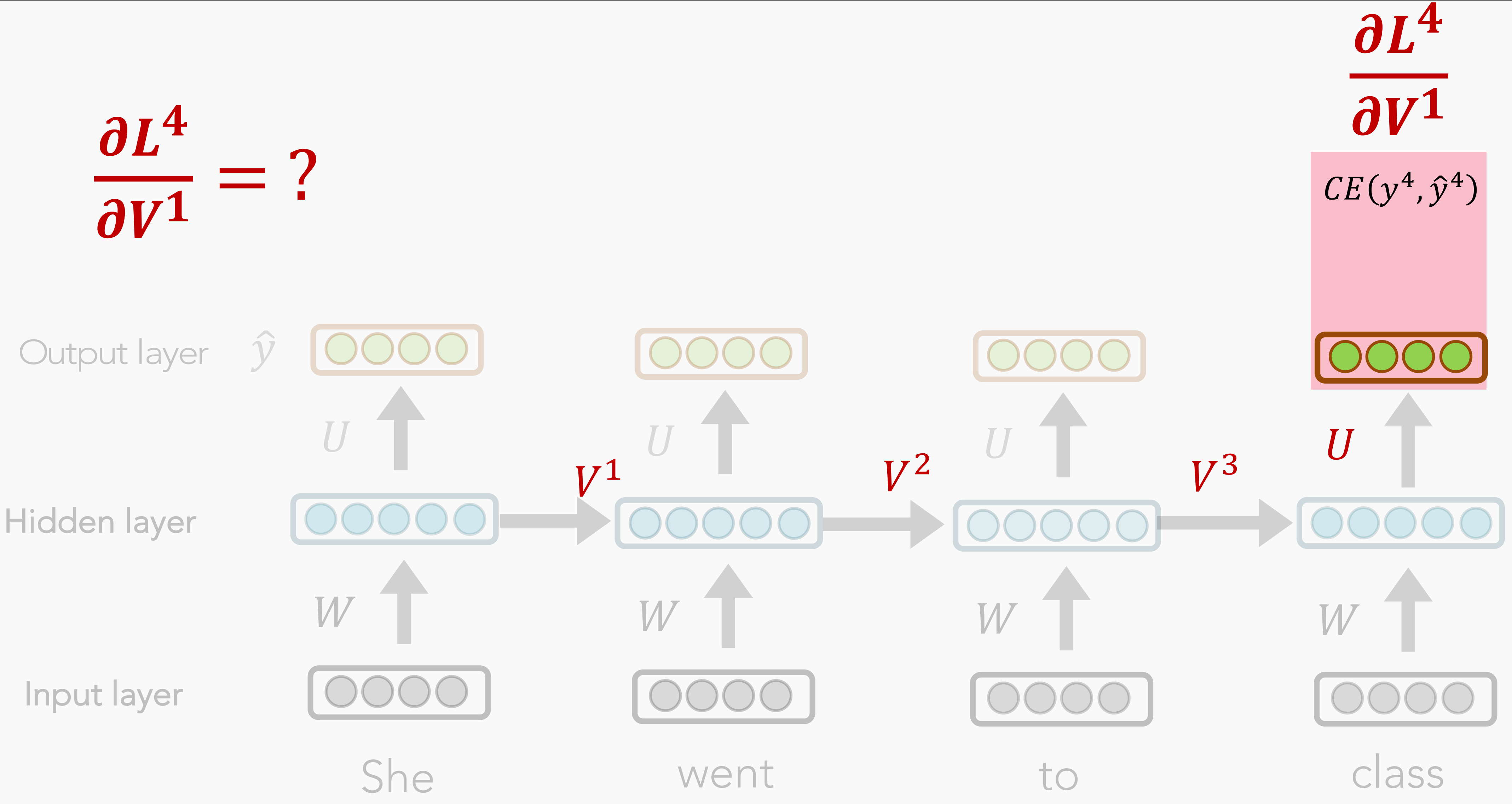
- Can handle infinite-length sequences (not just a fixed-window)
- Has a "memory" of the context (thanks to the hidden layer's recurrent loop)
- Same weights used for all inputs, so word order isn't wonky (like FFNN)

RNN ISSUES?

- Slow to train (BPTT)
- Due to "infinite sequence", gradients can easily **vanish** or **explode**
- Has trouble actually making use of long-range context

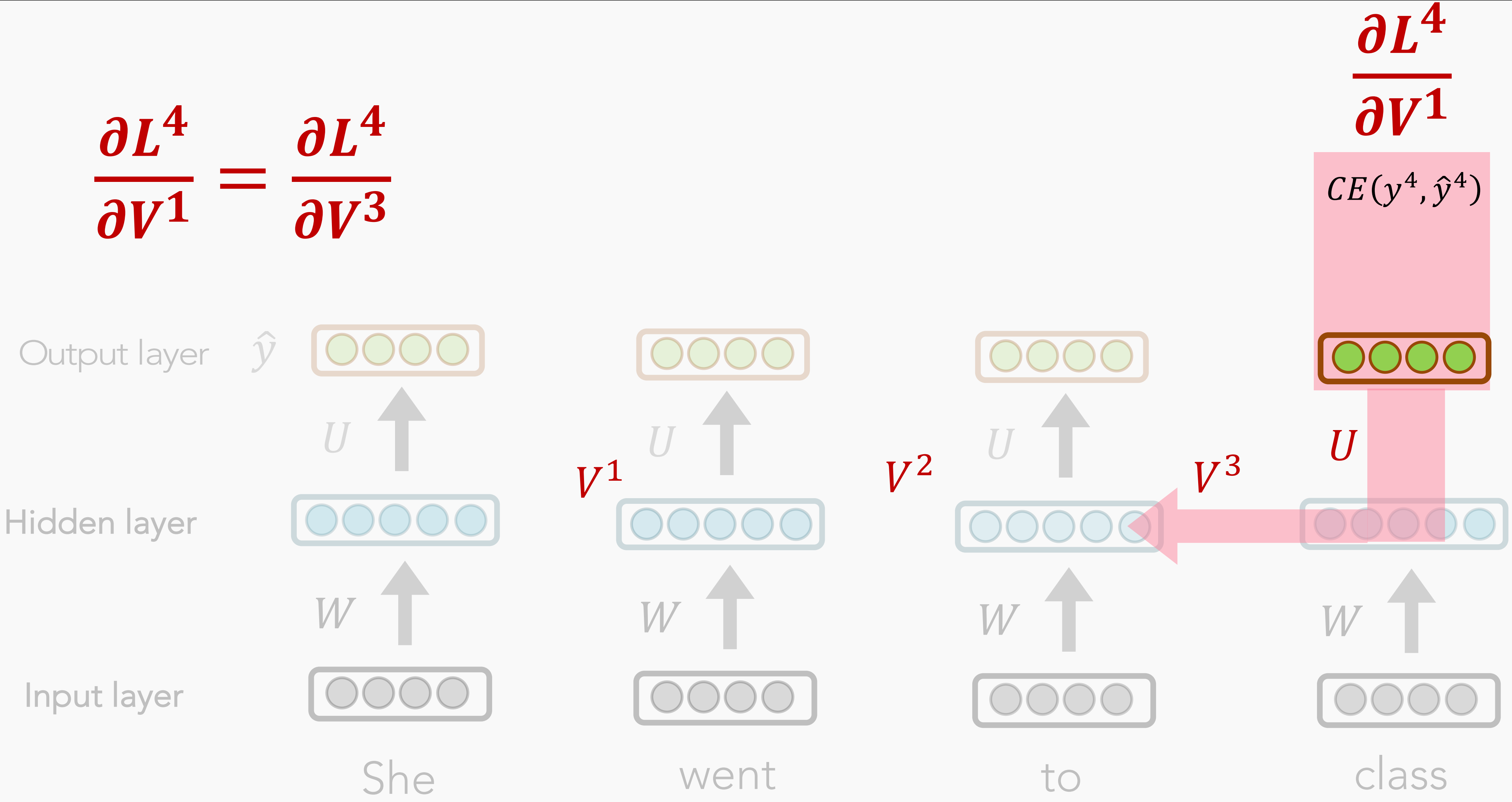
RNNs: Vanishing and Exploding Gradients (review)

$$\frac{\partial L^4}{\partial V^1} = ?$$



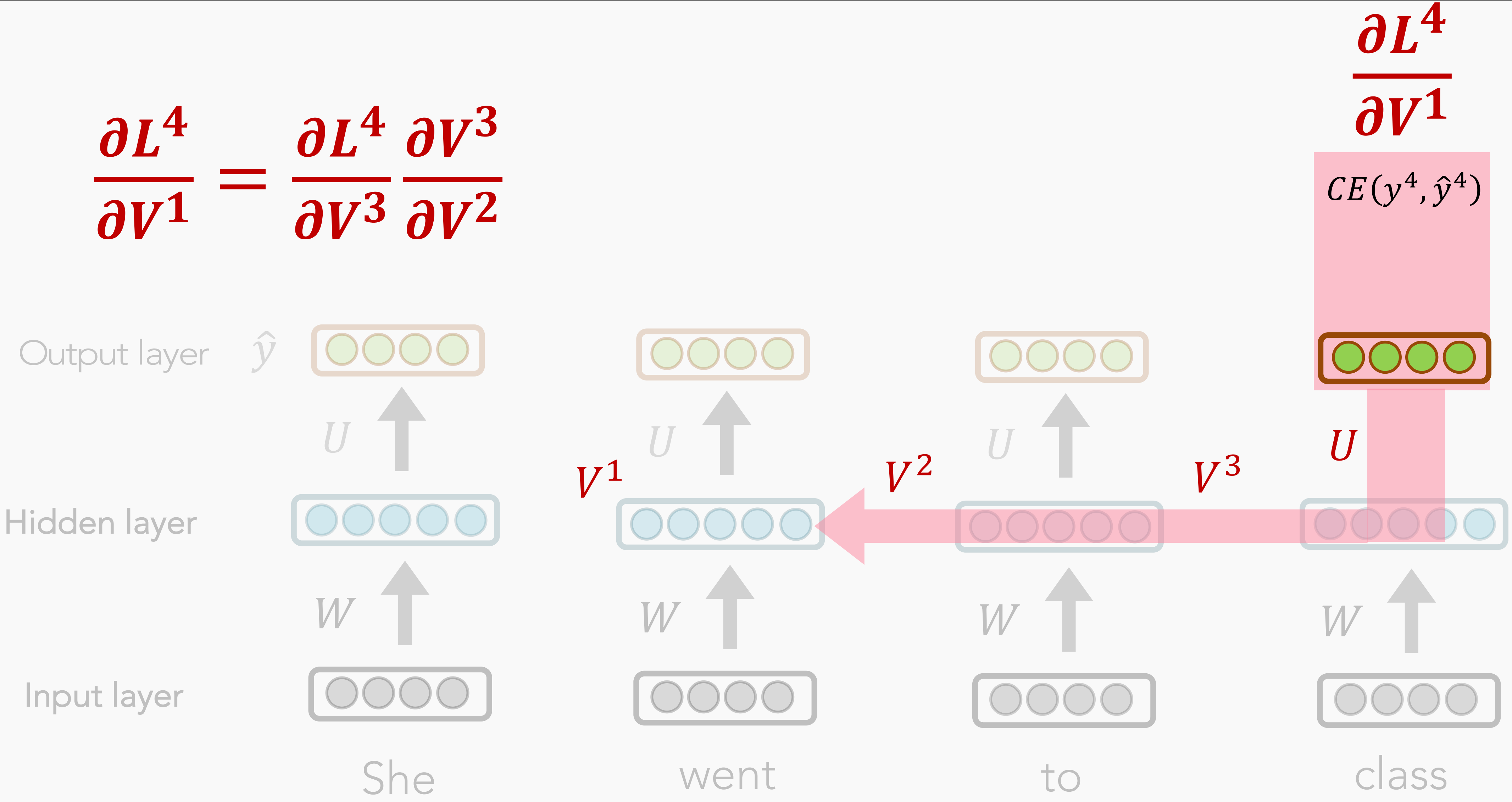
RNNs: Vanishing and Exploding Gradients (review)

$$\frac{\partial L^4}{\partial V^1} = \frac{\partial L^4}{\partial V^3}$$



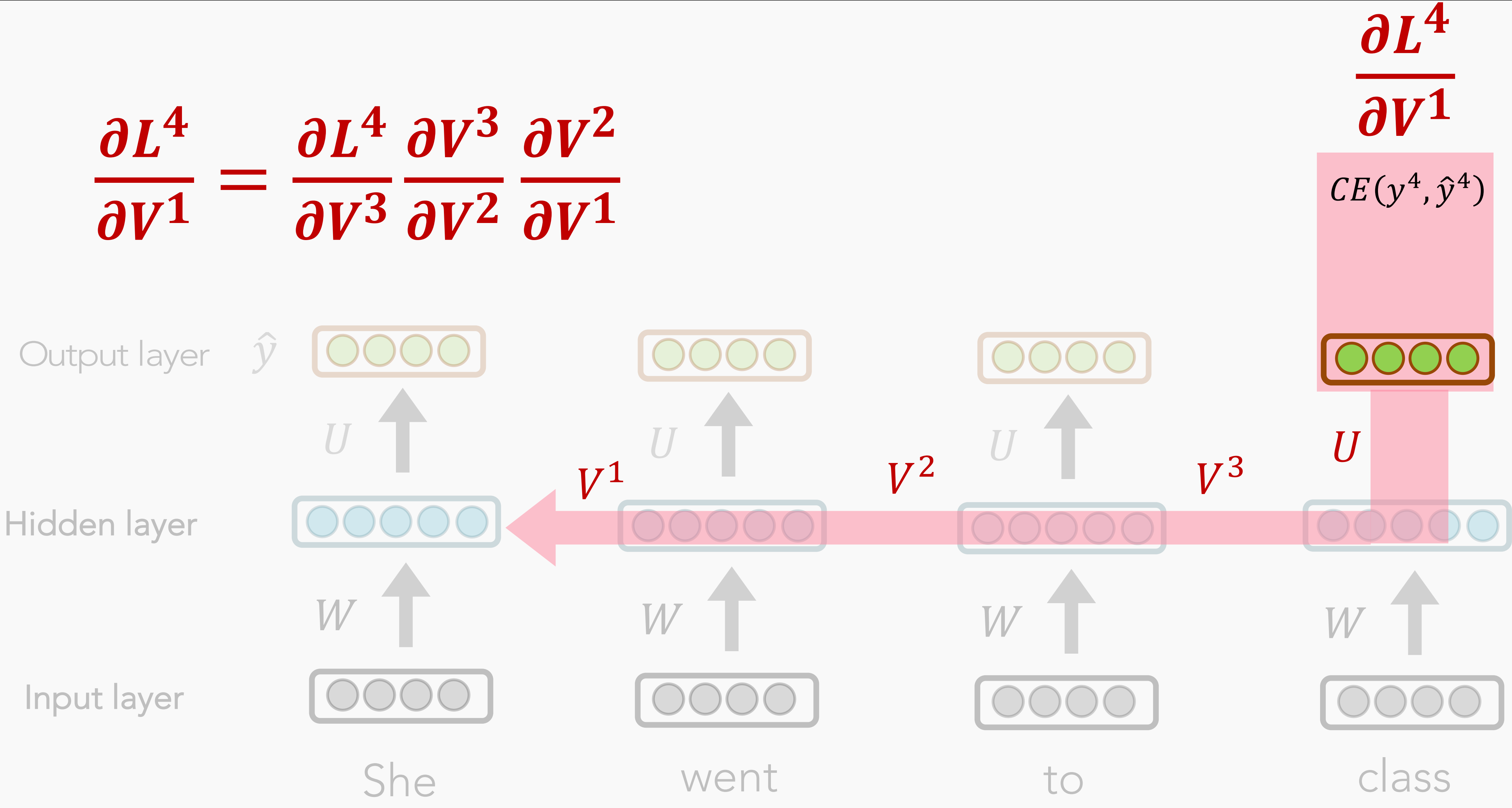
RNNs: Vanishing and Exploding Gradients (review)

$$\frac{\partial L^4}{\partial V^1} = \frac{\partial L^4}{\partial V^3} \frac{\partial V^3}{\partial V^2}$$



RNNs: Vanishing and Exploding Gradients (review)

$$\frac{\partial L^4}{\partial V^1} = \frac{\partial L^4}{\partial V^3}\frac{\partial V^3}{\partial V^2}\frac{\partial V^2}{\partial V^1}$$



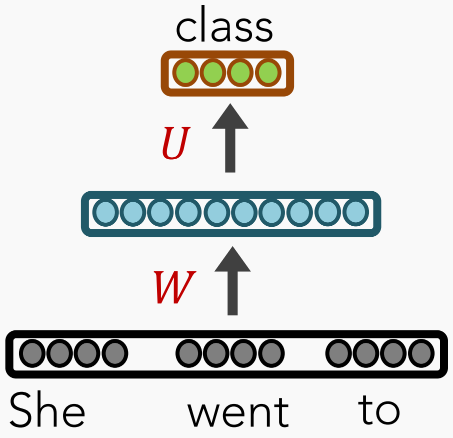
RNNs: Vanishing and Exploding Gradients (review)

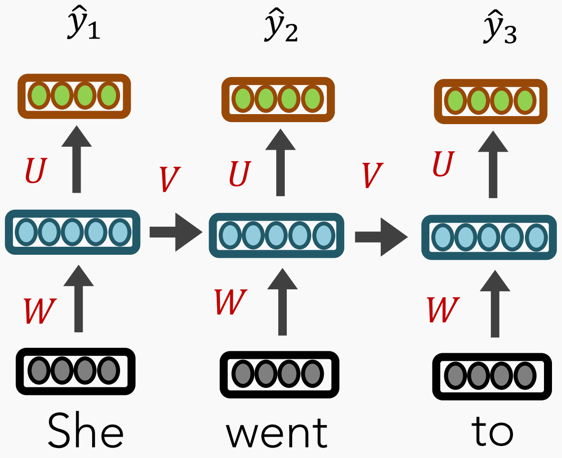
To address RNNs' finnicky nature with long-range context, we turned to an RNN variant named **LSTMs (long short-term memory)**

But first, let's recap what we've learned so far

Sequential Modelling (so far)

$$P(\text{went}|She) = \frac{\text{count}(She\ went)}{\text{count}(She)}$$





n-grams

- Basic counts; fast
- Fixed window size
- Sparsity & storage issues
- Not robust

FFNN

- Kind of robust… almost
- Fixed window size
- Weirdly handles context positions
- No "memory" of past

RNN 

- Handles infinite context (in theory)
- Robust to rare words
- Slow
- Difficulty with long context

Outline

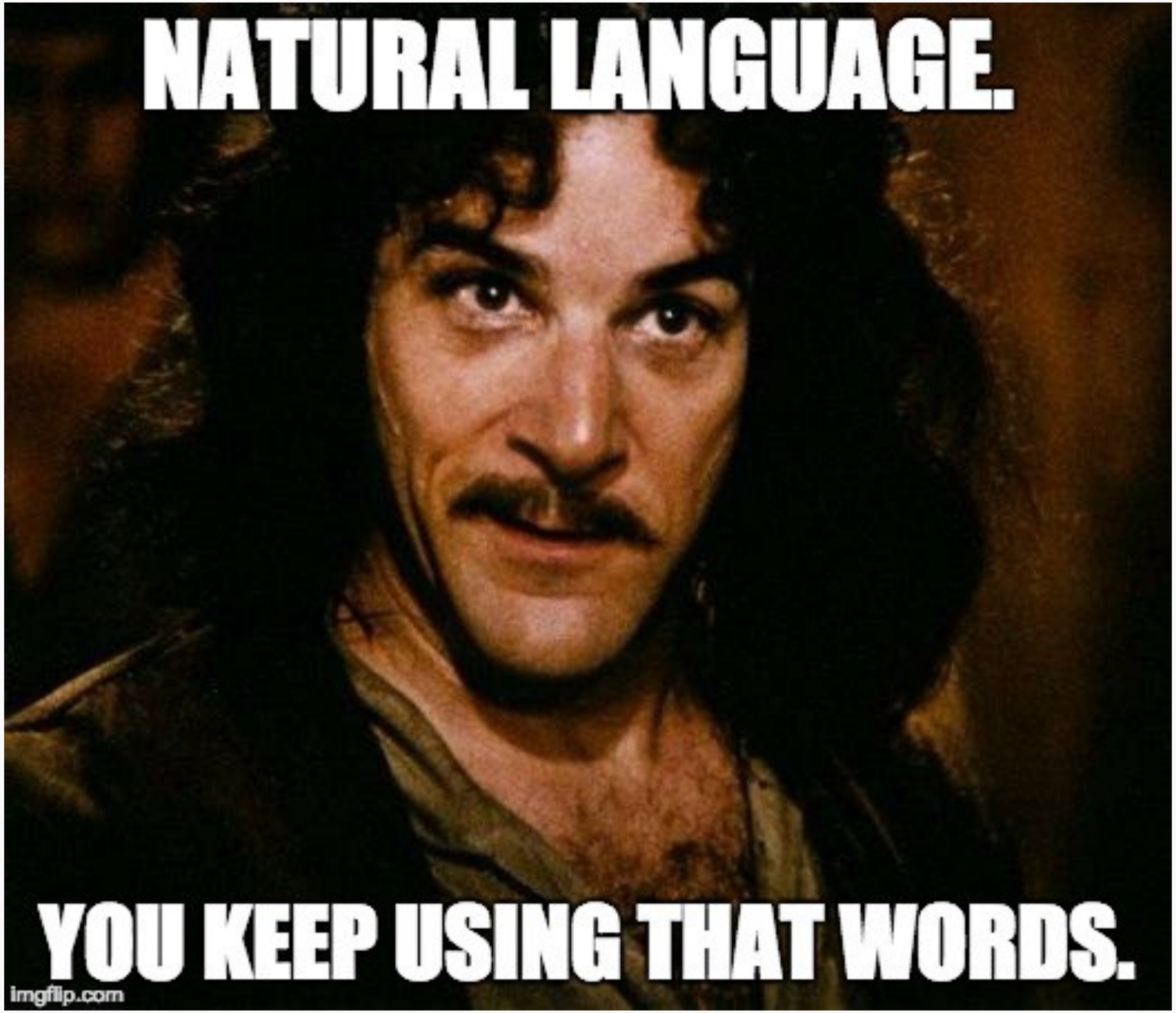
Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

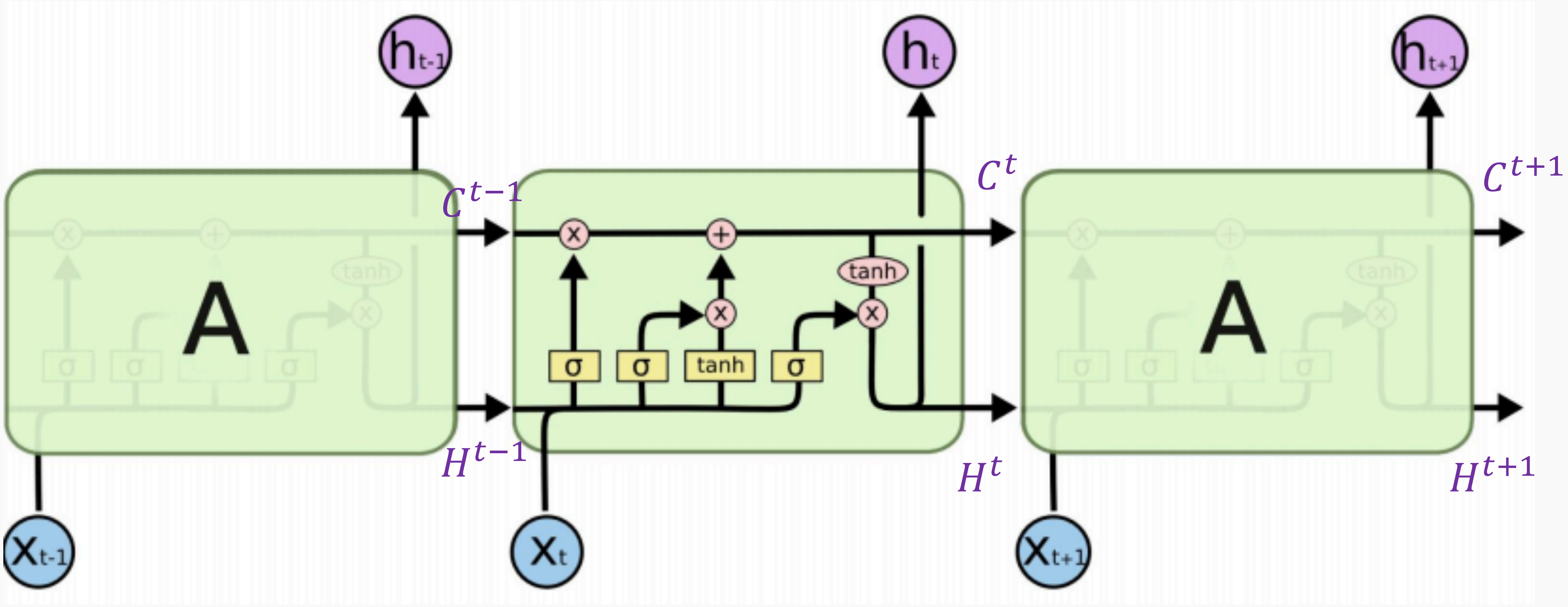
Conclusions


NATURAL LANGUAGE.
YOU KEEP USING THAT WORDS.
imgflip.com

Long short-term memory (LSTM)

- A type of RNN that is designed to better handle **long-range dependencies**
- In ”vanilla” RNNs, the hidden state is perpetually being rewritten
- In addition to a traditional **hidden state h**, let’s have a dedicated **memory cell c** for long-term events. More power to relay sequence info.

Inside an LSTM Hidden Layer



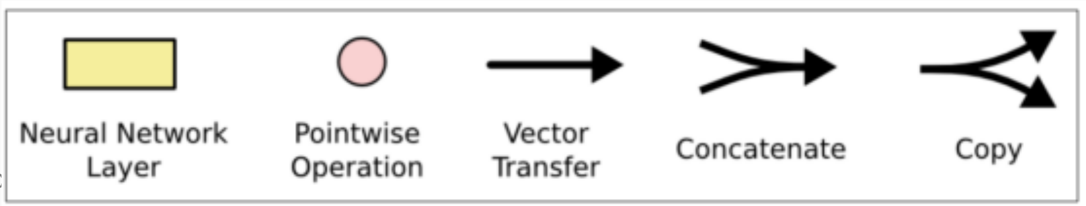


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

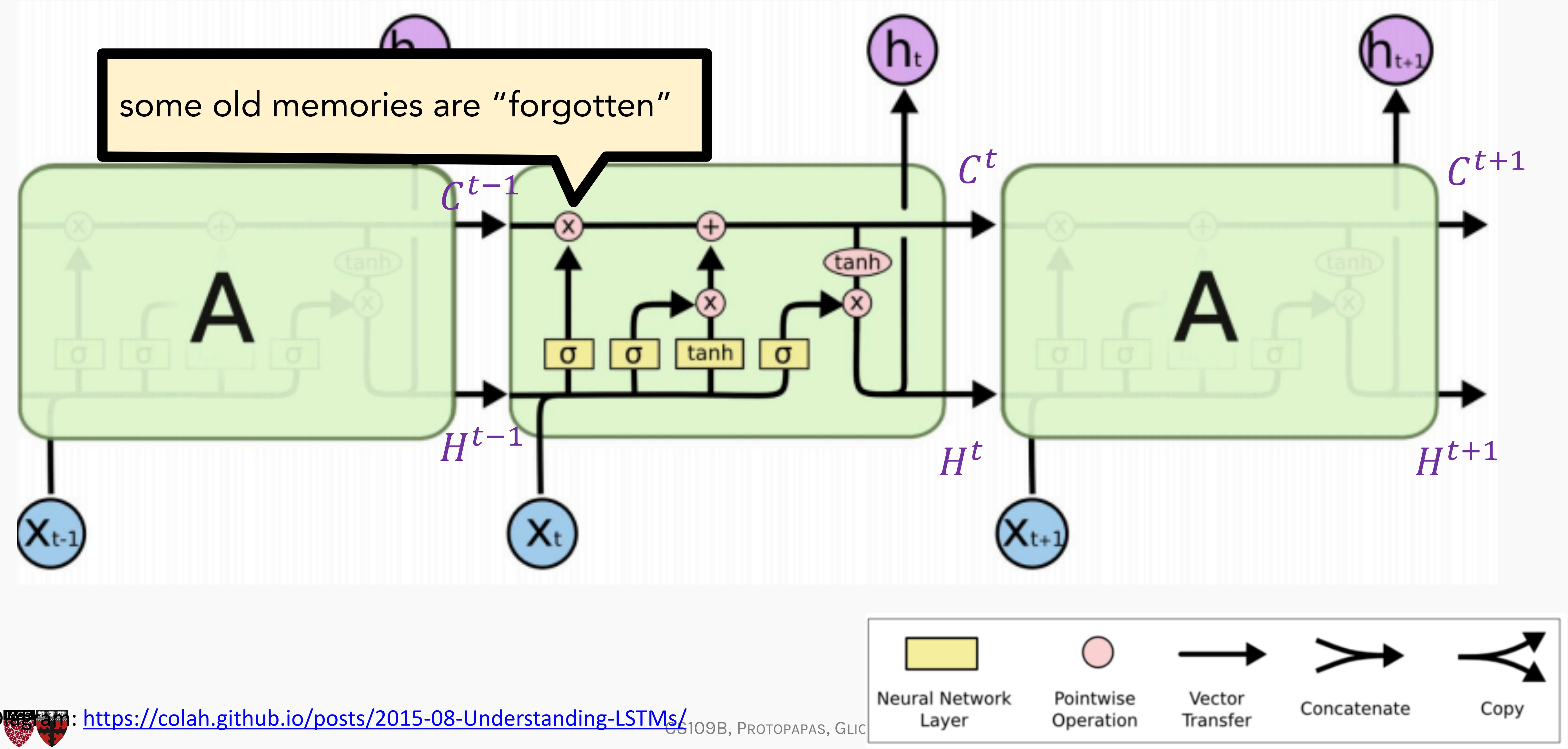


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

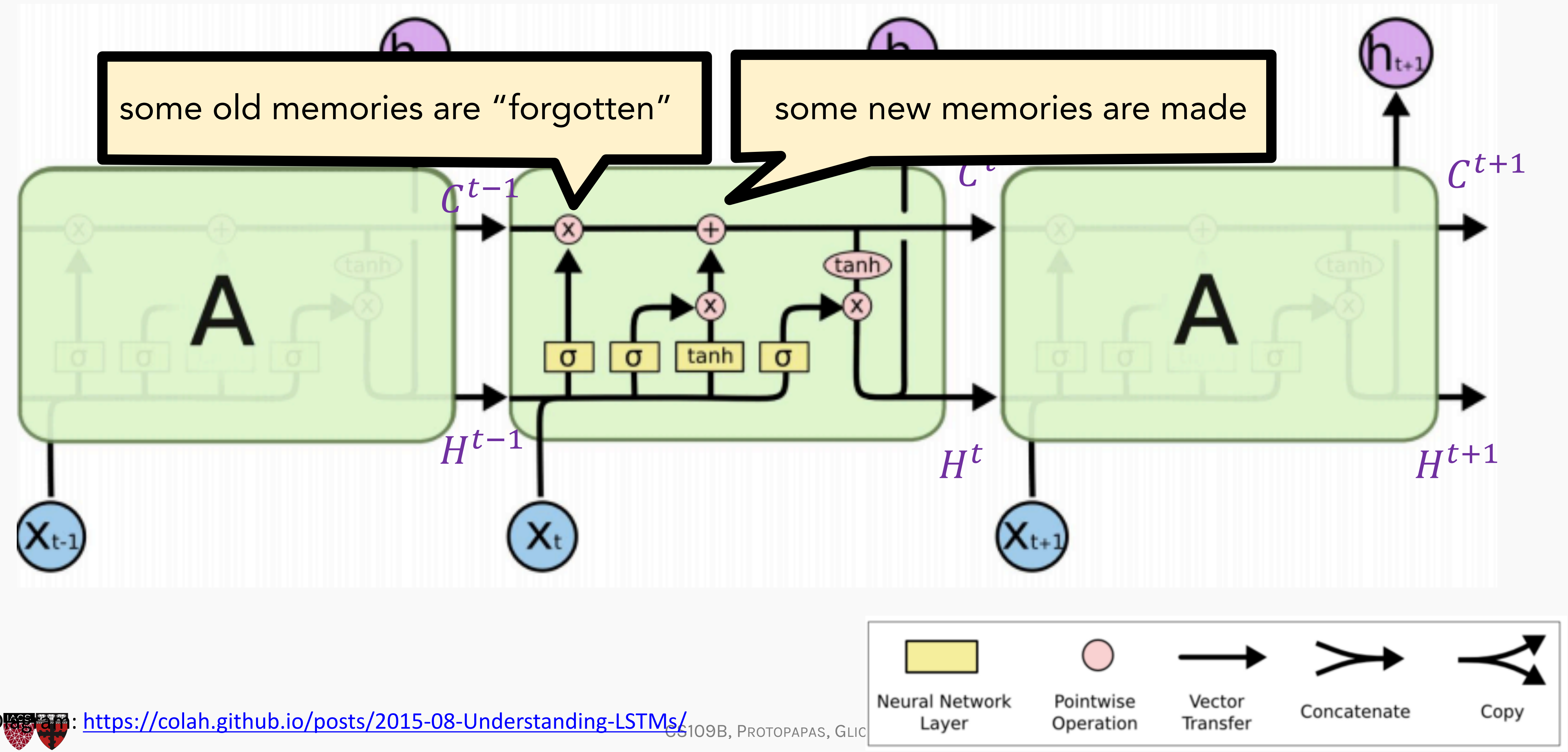


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

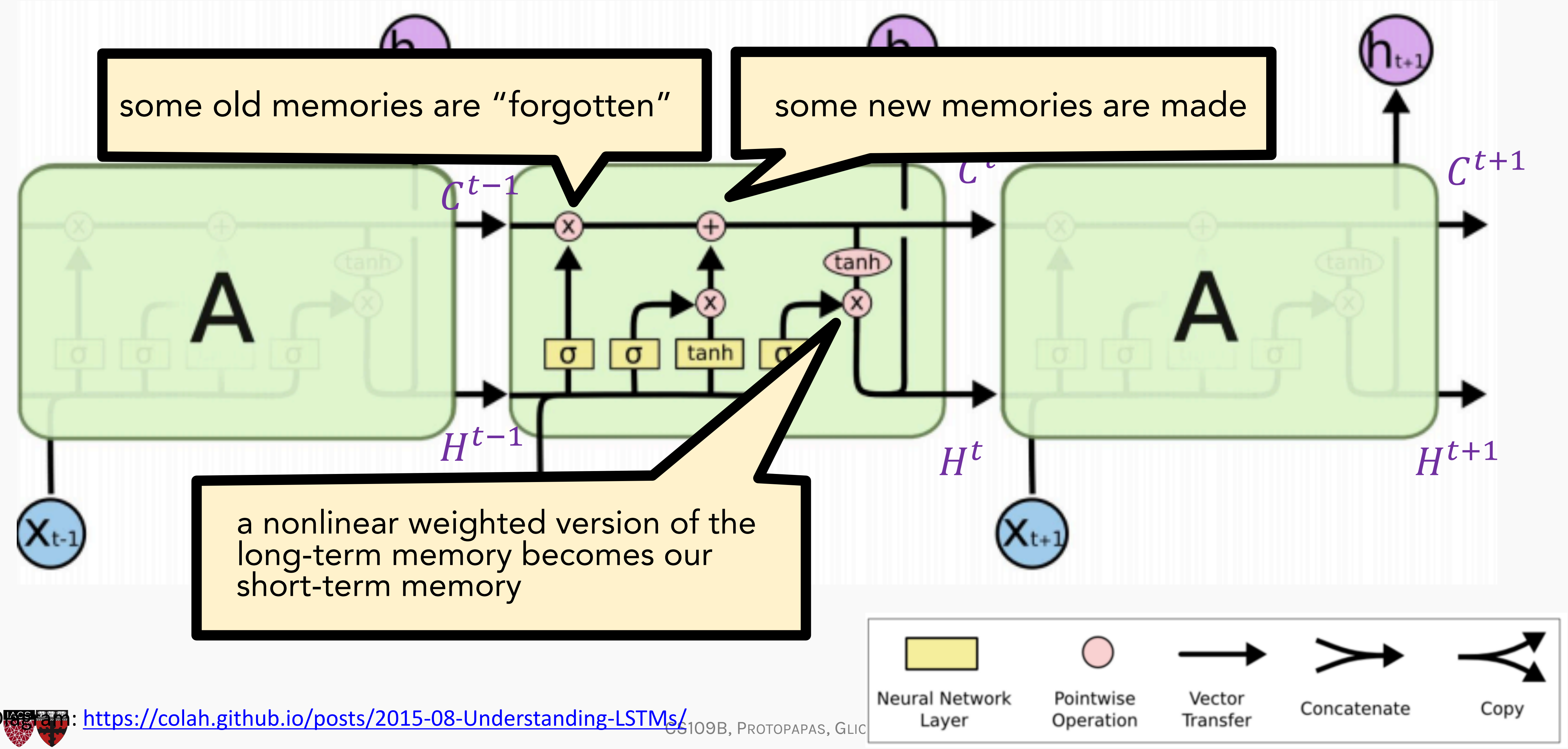
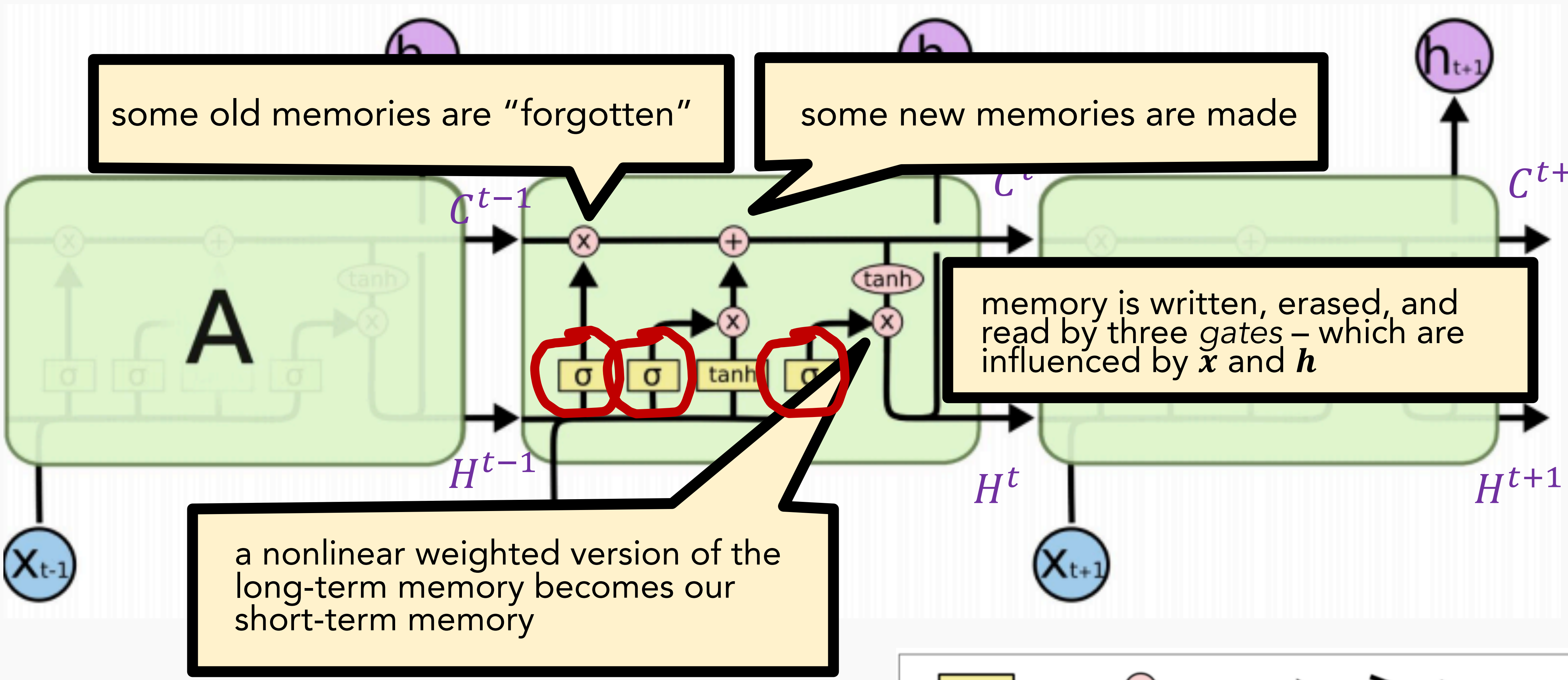


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer



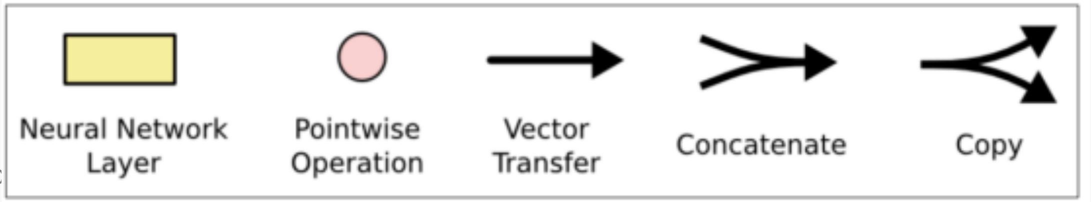


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inside an LSTM Hidden Layer

It's still possible for LSTMs to suffer from vanishing/exploding gradients, but it's way less likely than with vanilla RNNs:

- If RNNs wish to preserve info over long contexts, it must delicately find a recurrent weight matrix W_h that isn't too large or small

- However, LSTMs have 3 separate mechanism that adjust the flow of information (e.g., forget gate, if turned off, will preserve all info)

Long short-term memory (LSTM)

LSTM STRENGTHS?

- Almost always outperforms vanilla RNNs
- Captures long-range dependencies shockingly well

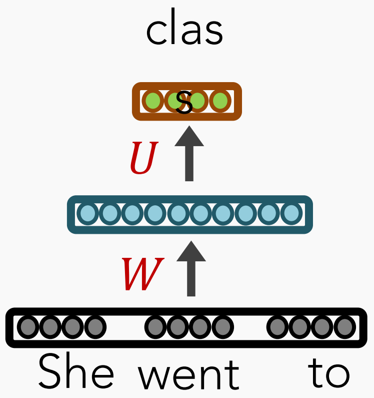
LSTM ISSUES?

- Has more weights to learn than vanilla RNNs; thus,
- Requires a moderate amount of training data (otherwise, vanilla RNNs are better)
- Can still suffer from vanishing/exploding gradients

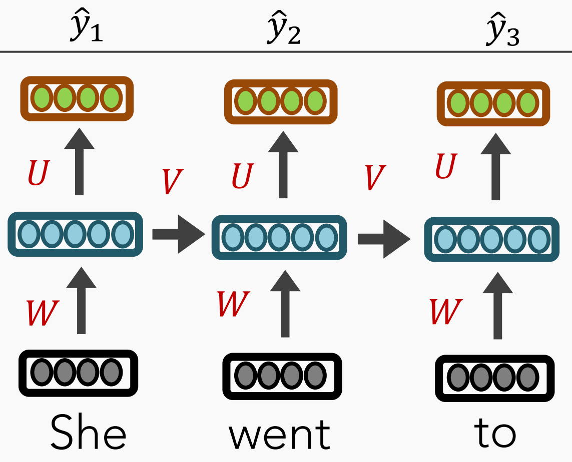
Sequential Modelling

$$P(\text{went}|She) = \frac{\text{count}(She\ went)}{\text{count}(She)}$$

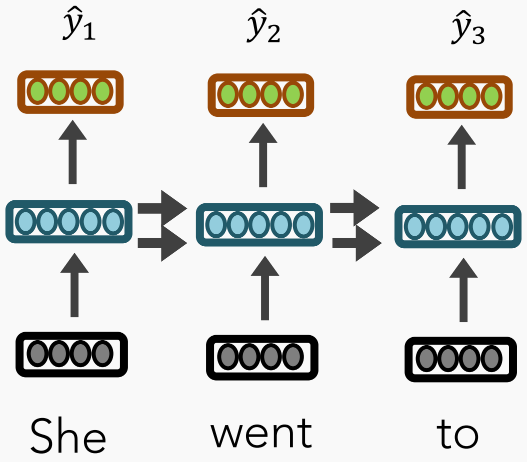
n-grams



FFNN



RNN 



LSTM 

Sequential Modelling

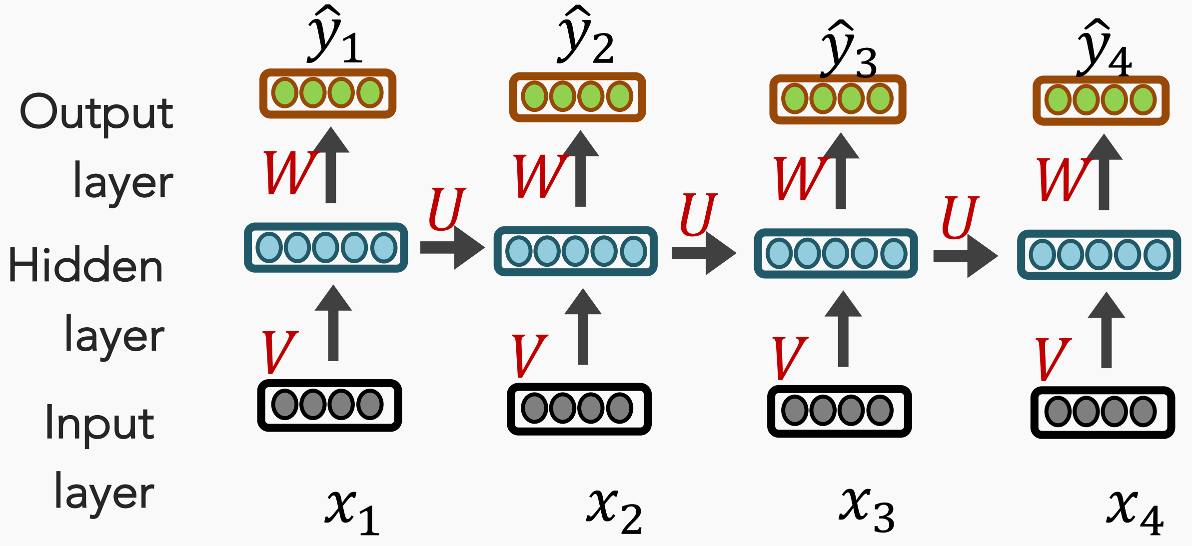
IMPORTANT

If your goal isn't to predict the next item in a sequence, and you rather do some other <u>classification or regression task</u> using the sequence, then you can:

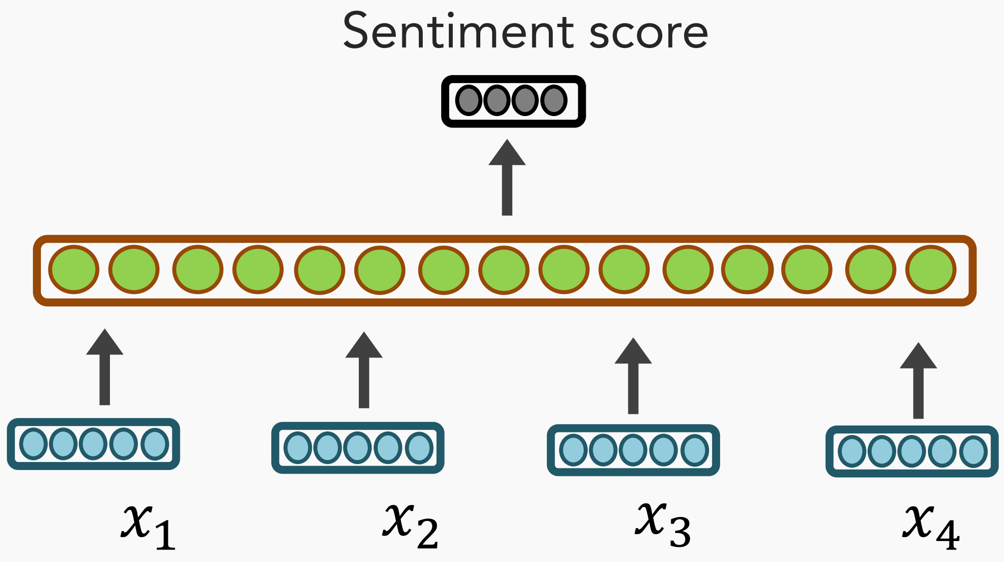
- Train an aforementioned model (e.g., LSTM) as a language model
- Use the **hidden layers** that correspond to each item in your sequence

Sequential Modelling

1. Train LM to learn hidden layer embeddings

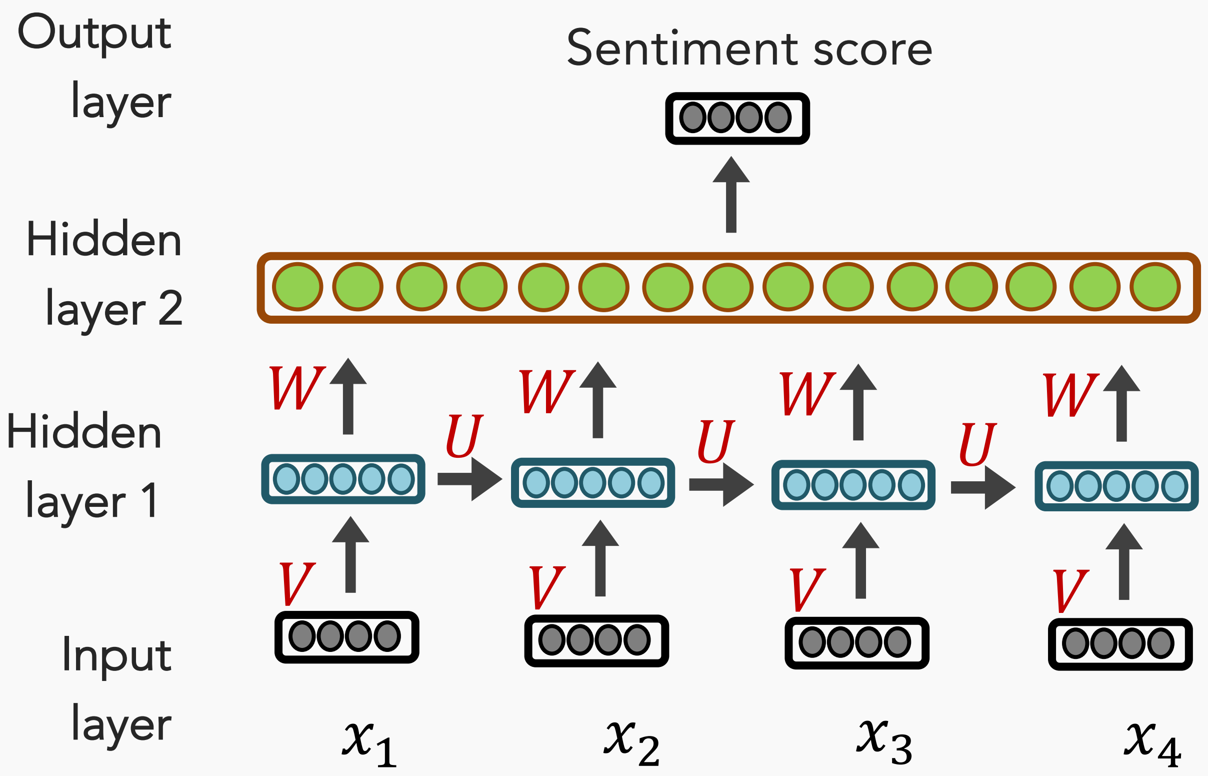


2. Use hidden layer embeddings for other tasks



Sequential Modelling

Or jointly learn hidden embeddings toward a particular task (end-to-end)



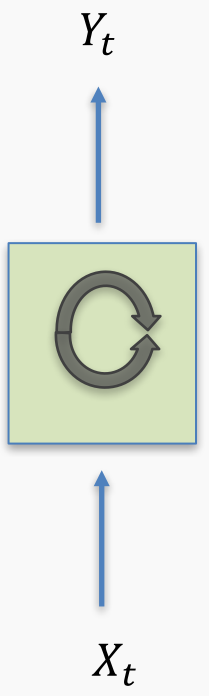
You now have the foundation for modelling sequential data.

Most state-of-the-art advances are based on those core RNN/LSTM ideas. But, with tens of thousands of researchers and hackers exploring deep learning, there are many tweaks that haven proven useful.

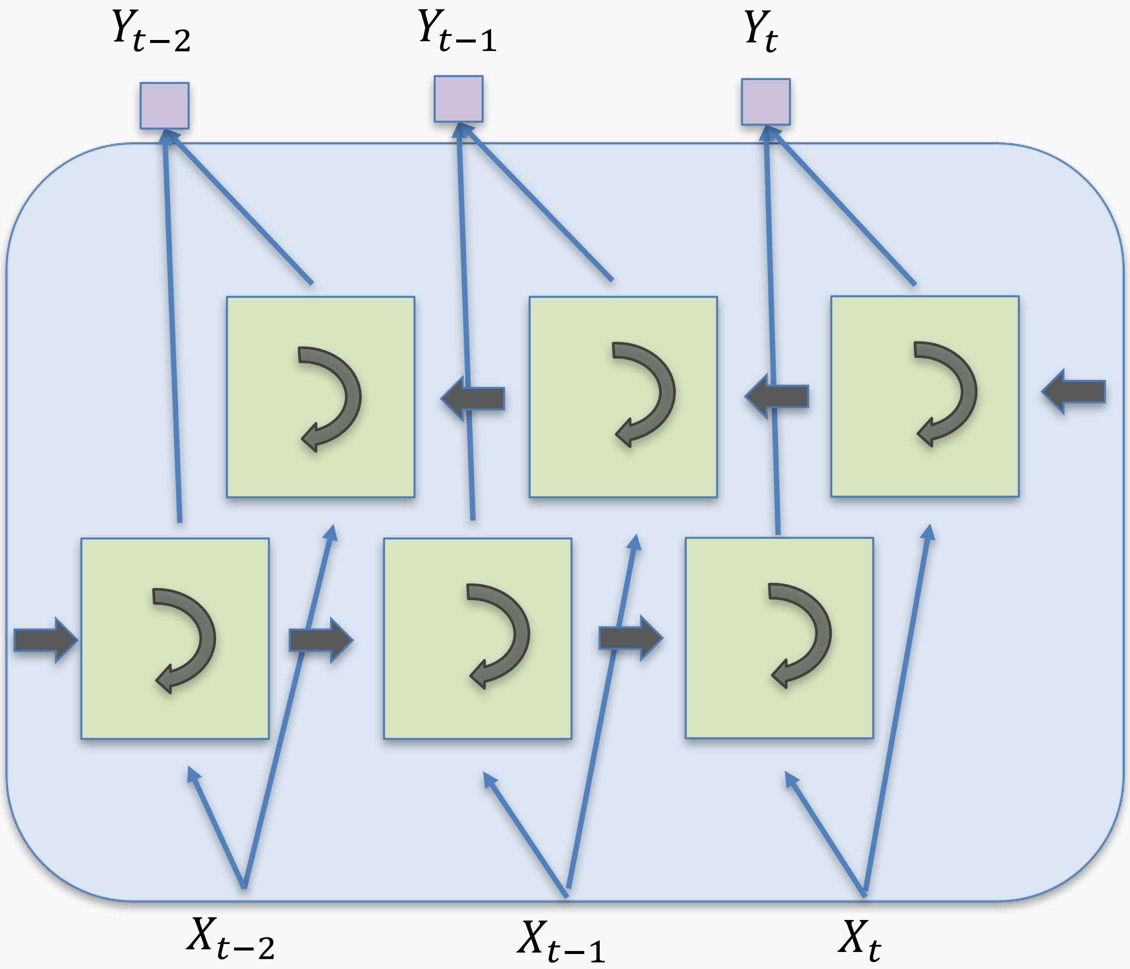
(This is where things get crazy.)

Bi-directional (review)

symbol for a BRNN



previous state



previous state

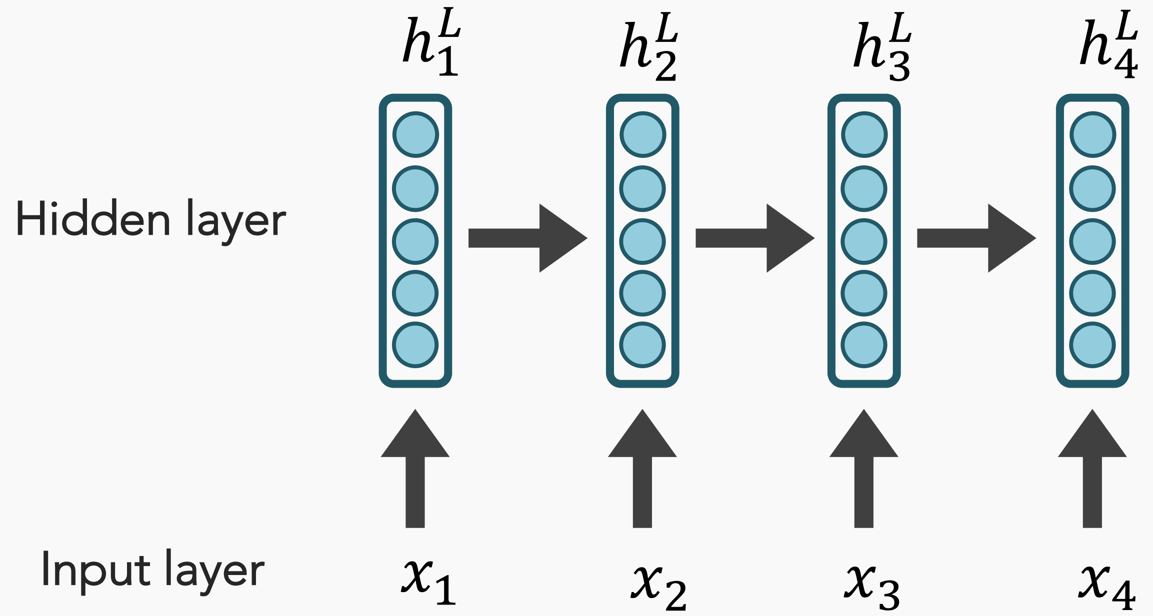
Bi-directional (review)

RNNs/LSTMs use the left-to-right context and sequentially process data.

If you have full access to the data at testing time, why not make use of the flow of information from right-to-left, also?

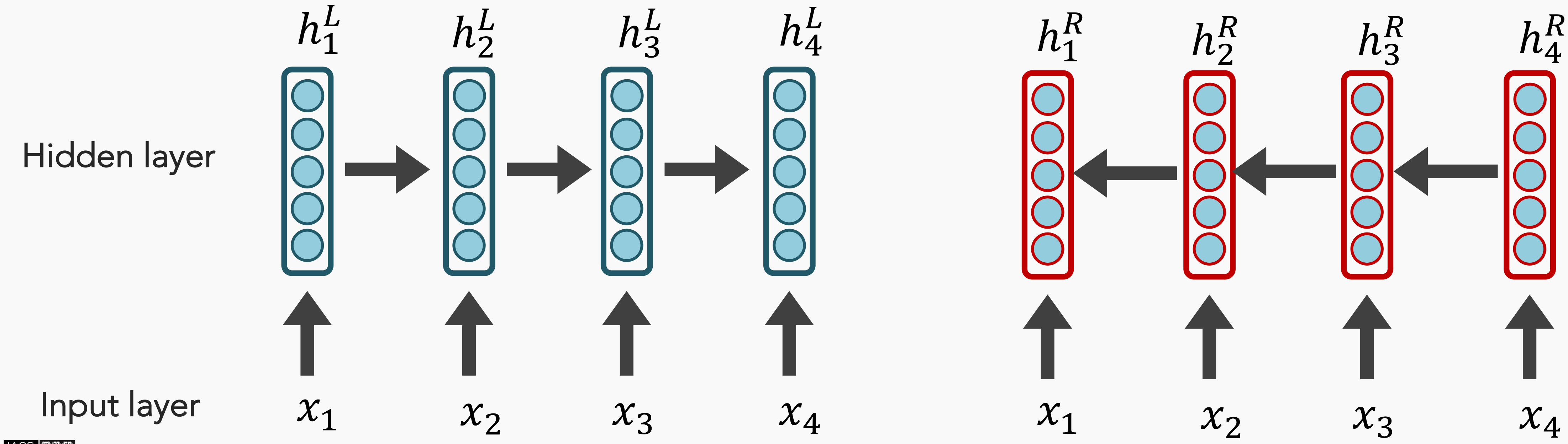
RNN Extensions: **Bi-directional LSTMs (review)**

For brevity, let's use the follow schematic to represent an RNN

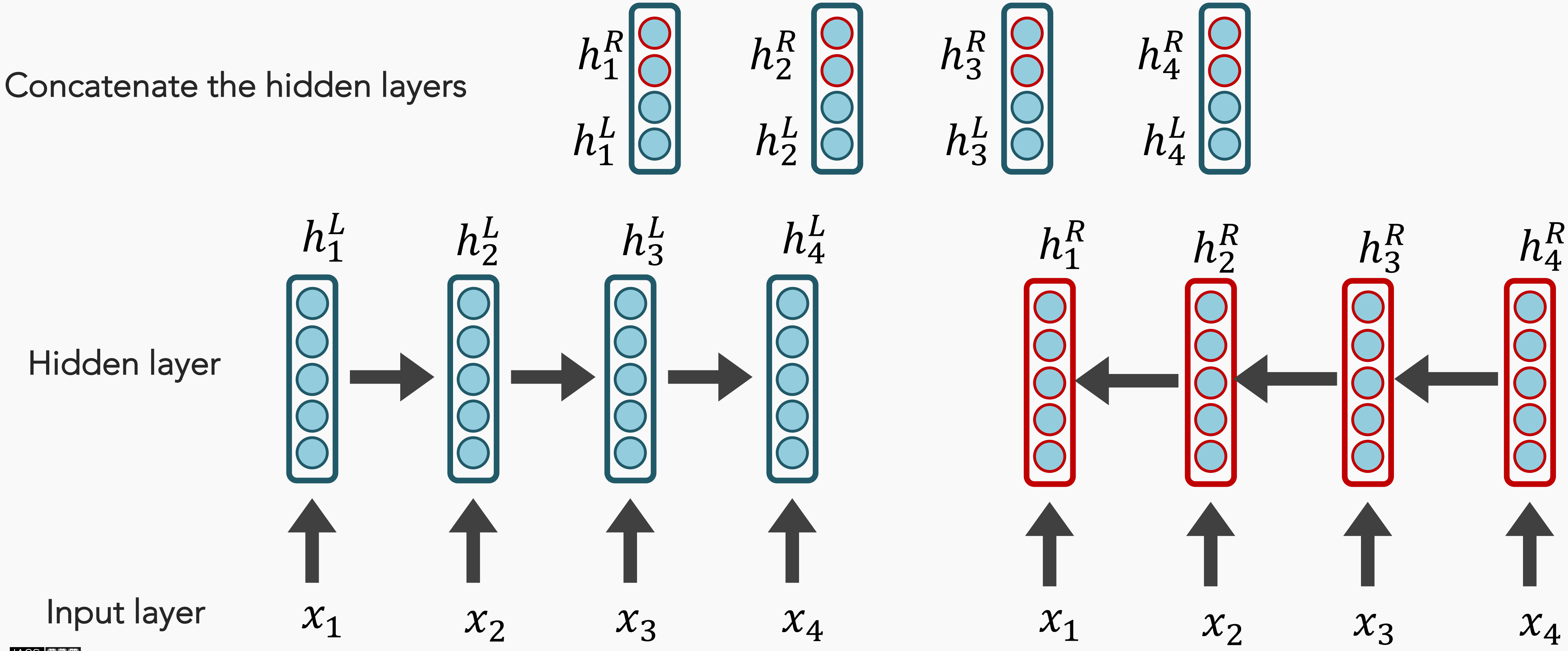


RNN Extensions: **Bi-directional LSTMs (review)**

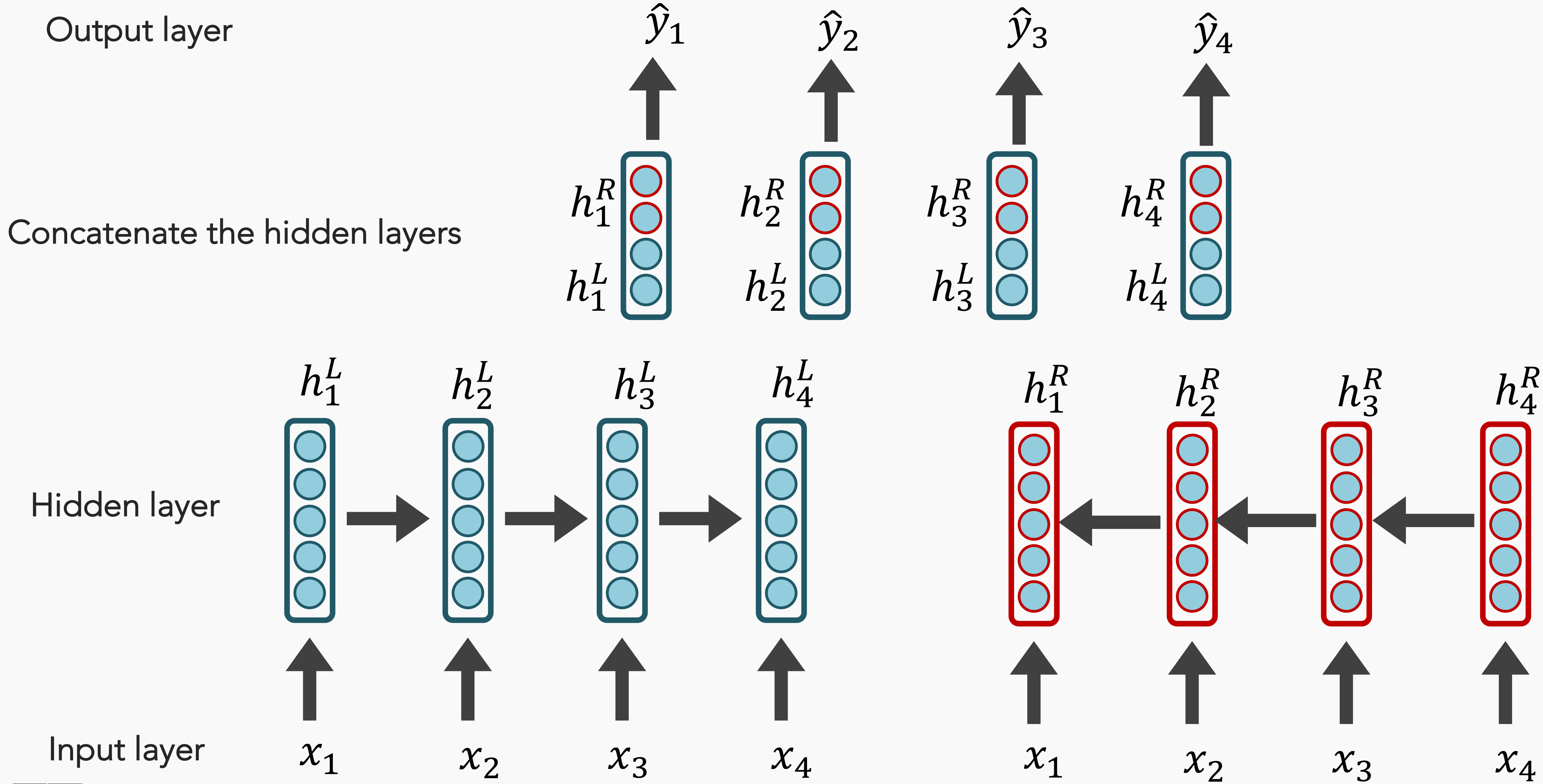
For brevity, let's use the follow schematic to represent an RNN



RNN Extensions: **Bi-directional LSTMs (review)**



RNN Extensions: **Bi-directional LSTMs (review)**



RNN Extensions: **Bi-directional LSTMs (review)**

BI-LSTM STRENGTHS?

- Usually performs at least as well as uni-directional RNNs/LSTMs

BI-LSTM ISSUES?

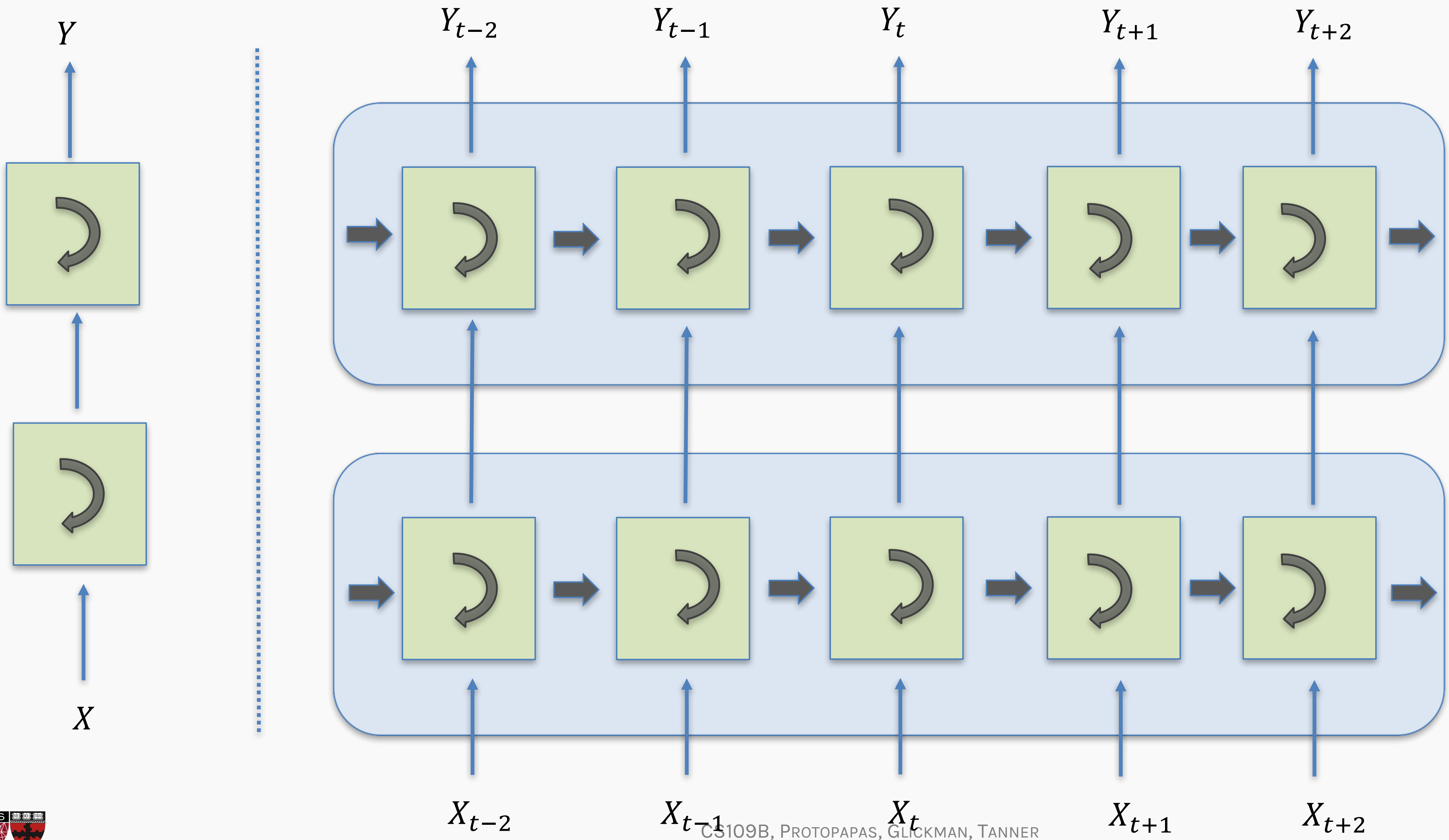
- Slower to train
- Only possible if access to full data is allowed

Deep RNN (review)

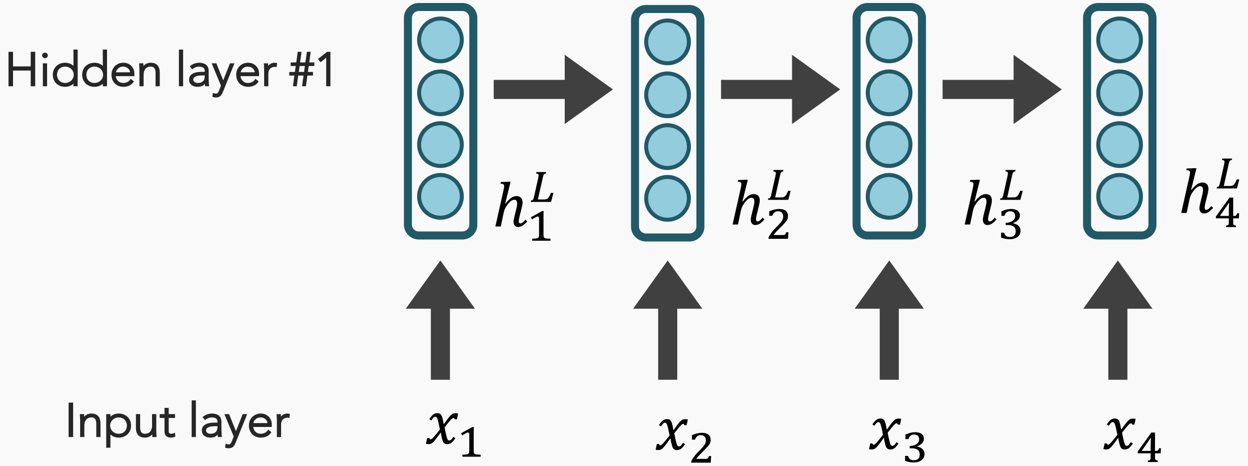
LSTMs units can be arranged in layers, so that the output of each unit is the input to the other units. This is called **a deep RNN**, where the adjective “deep” refers to these multiple layers.

- Each layer feeds the LSTM on the next layer
- First time step of a feature is fed to the first LSTM, which processes that data and produces an output (and a new state for itself).
- That output is fed to the next LSTM, which does the same thing, and the next, and so on.
- Then the second time step arrives at the first LSTM, and the process repeats.

Deep RNN (review)



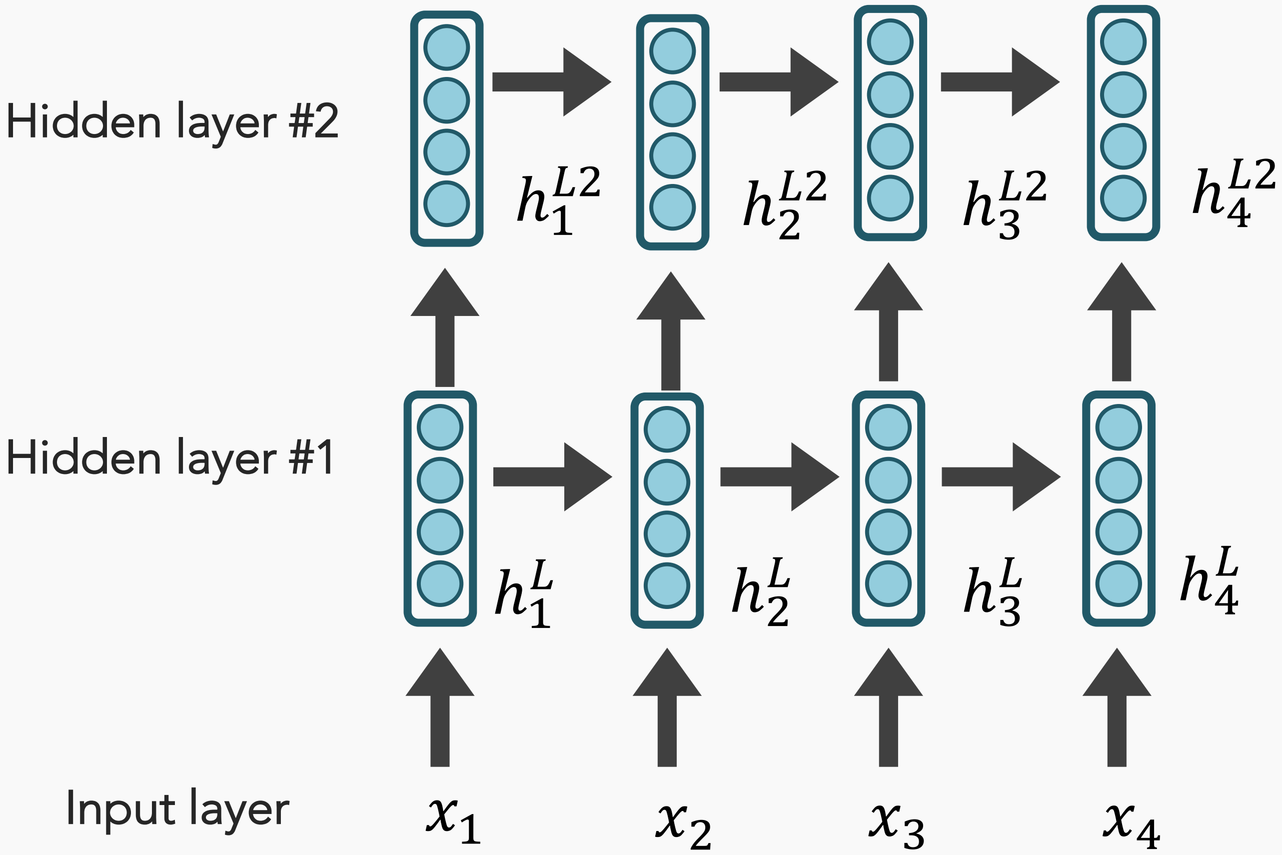
Deep RNN (review)



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

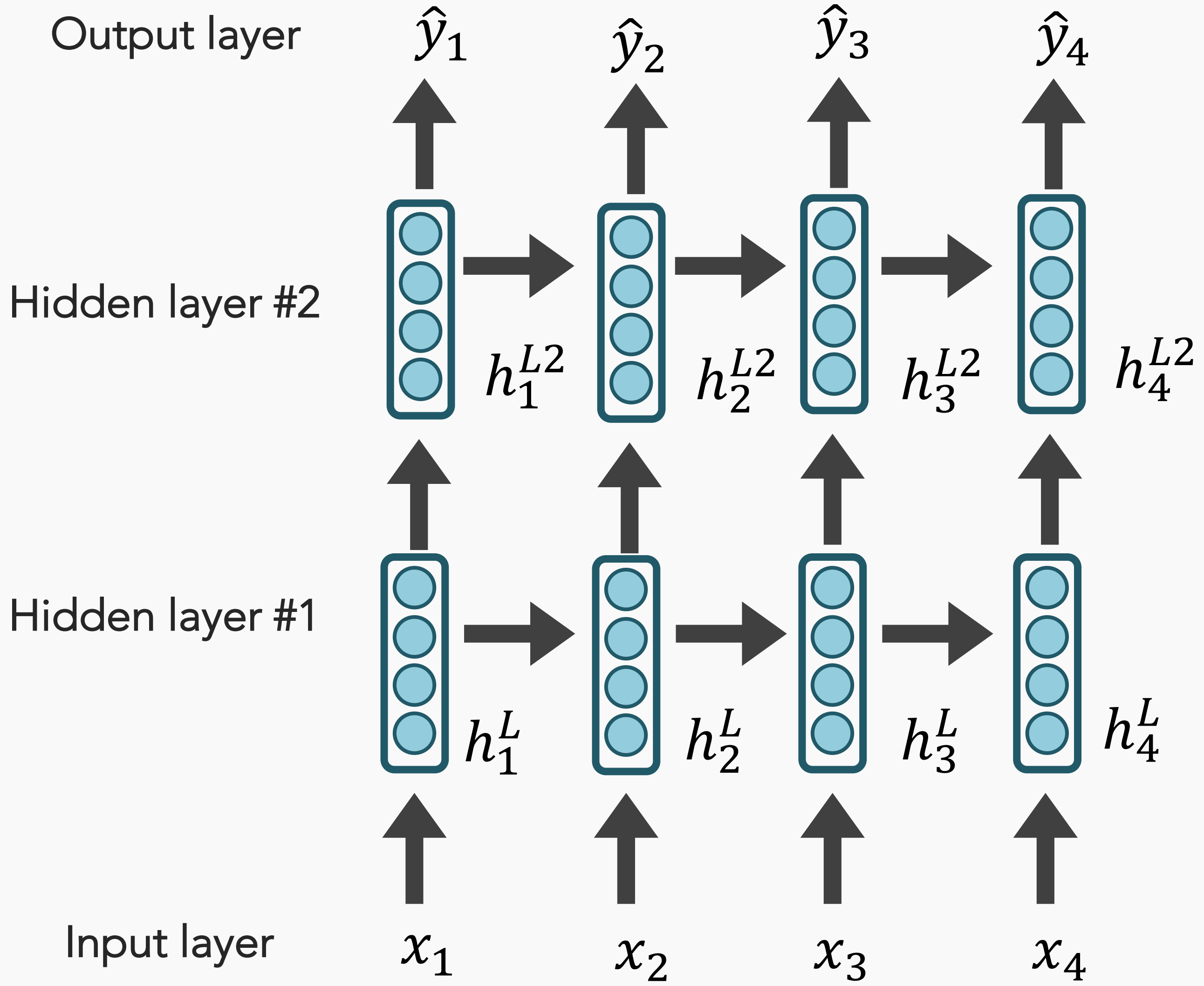
Deep RNN (review)



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

Deep RNN (review)



Hidden layers provide an abstraction (holds "meaning").

Stacking hidden layers provides increased abstractions.

ELMo: **Stacked Bi-directional LSTMs**

General Idea:

- Goal is to get highly rich embeddings for each word (unique type)
- Use both directions of context (bi-directional), with increasing abstractions (stacked)
- Linearly combine all abstract representations (hidden layers) and optimize w.r.t. a particular task (e.g., sentiment classification)

ELMo Slides: https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

ELMo: **Stacked Bi-directional LSTMs**

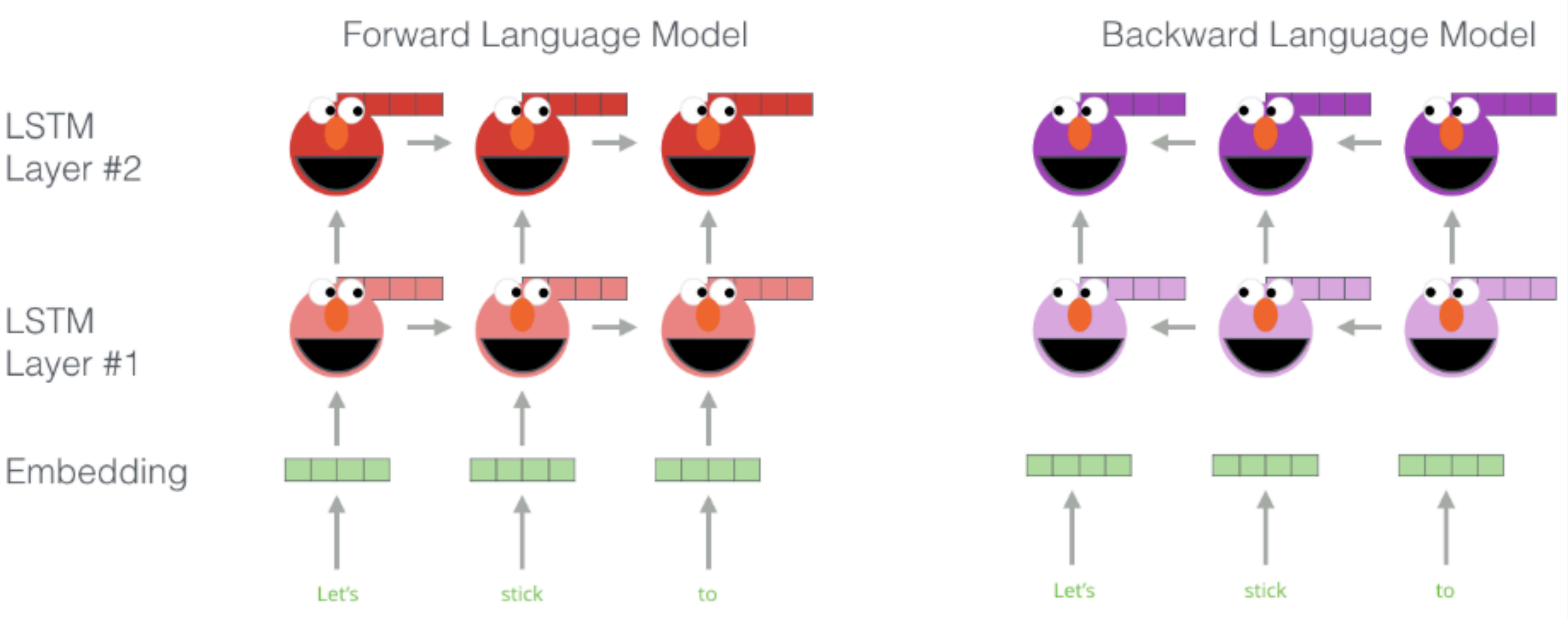


Illustration: http://jalammar.github.io/illustrated-bert/

Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

x s_2

x s_1

x s_0

3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context

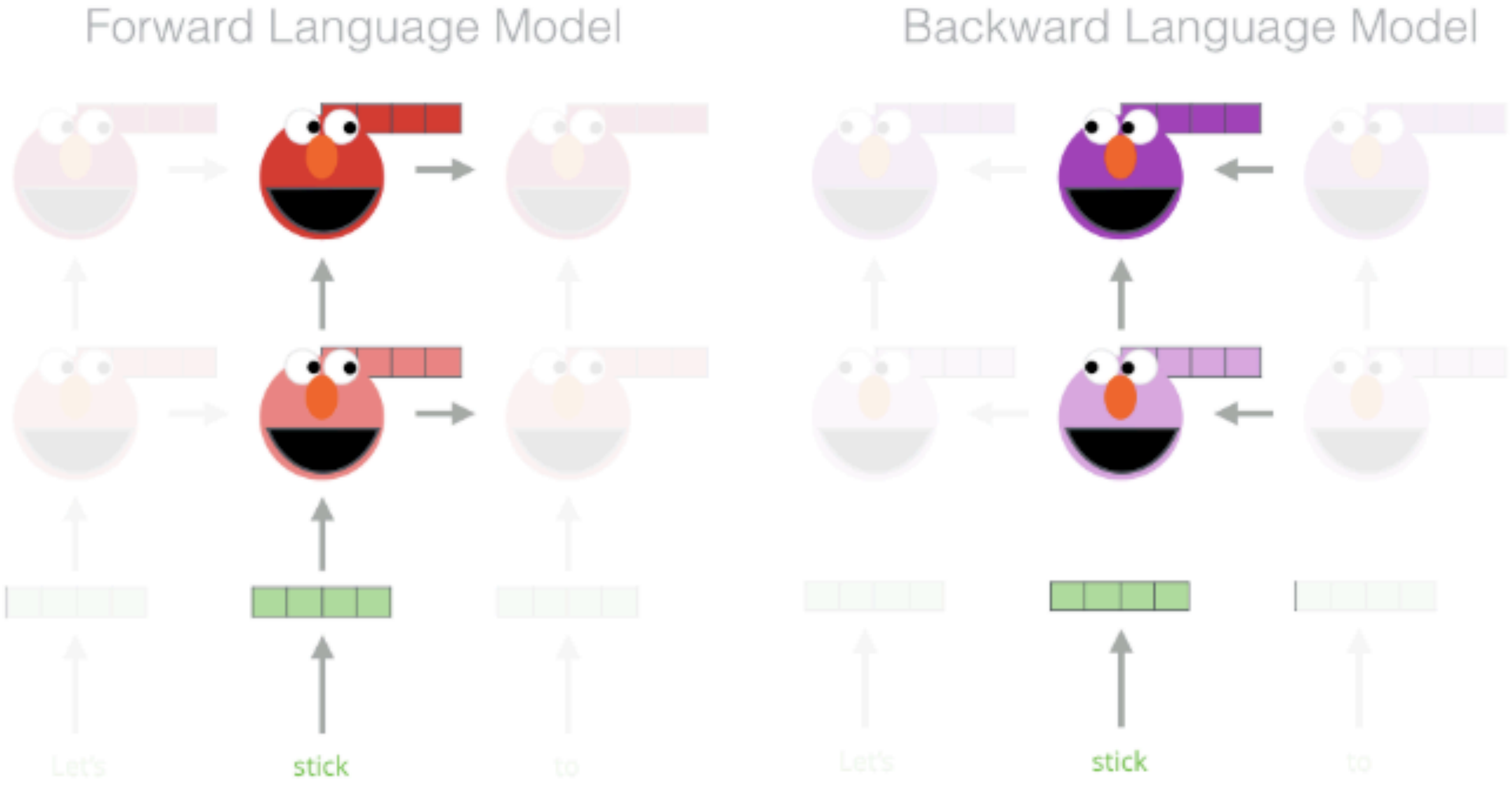


Illustration: http://jalammar.github.io/illustrated-bert/

ELMo: **Stacked Bi-directional LSTMs**

- ELMo yielded incredibly good word embeddings, which yielded state-of-the-art results when applied to many NLP tasks.

- Main ELMo takeaway: given enough training data, having tons of explicit connections between your vectors is useful
(system can determine how to best use context)

ELMo Slides: https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

REFLECTION

So far, for all of our sequential modelling, we have been concerned with emitting 1 output per input datum.

Sometimes, a *sequence* is the smallest granularity we care about though (e.g., an English sentence)

Outline

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

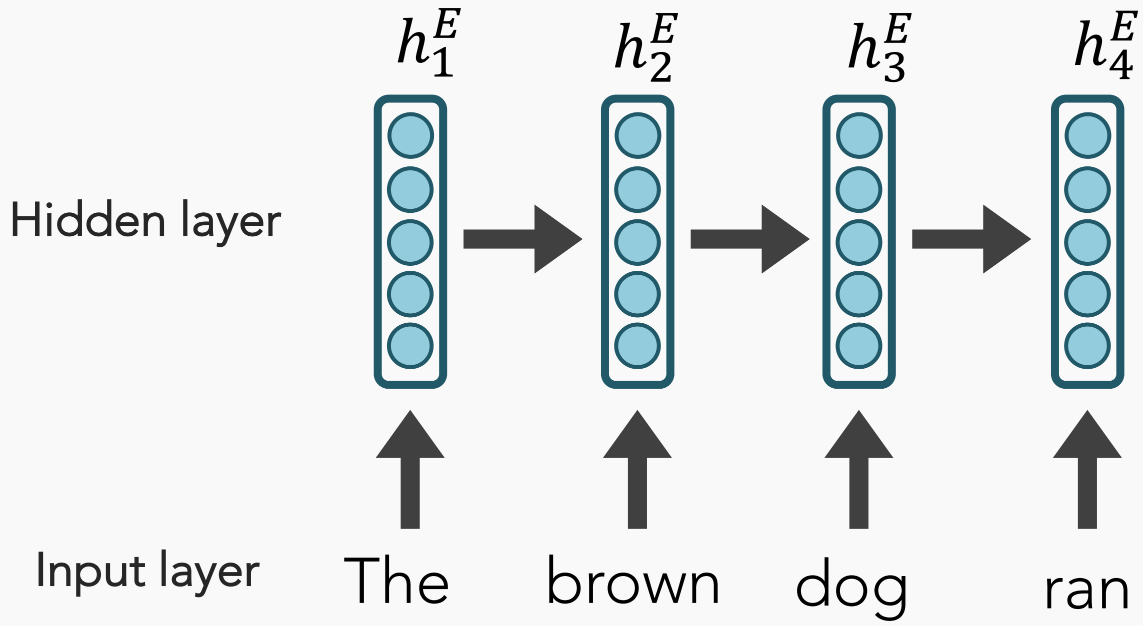
Conclusions


WAS THAT...
SARCASM?

Sequence-to-Sequence (seq2seq)

- If our input is a sentence in Language A, and we wish to translate it to Language B, it is clearly sub-optimal to translate word by word (like our current models are suited to do).
- Instead, let a ***sequence*** of tokens be the unit that we ultimately wish to work with (a sequence of length **N** may emit a sequences of length **M**)
- Seq2seq models are comprised of **2 RNNs**: 1 encoder, 1 decoder

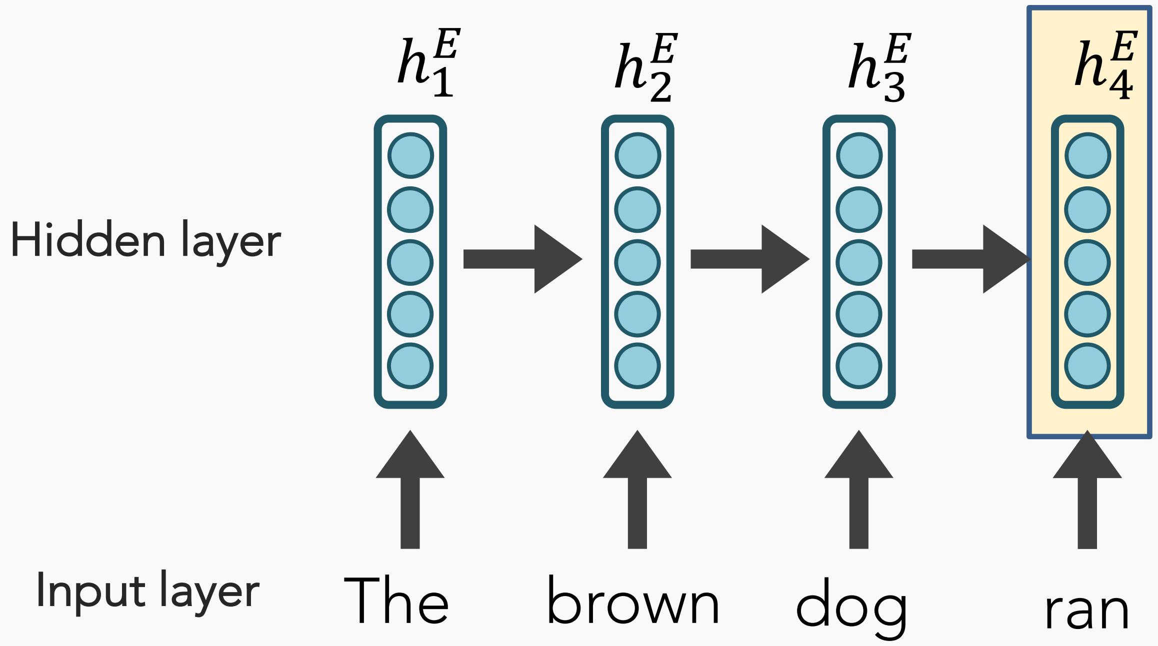
Sequence-to-Sequence (seq2seq)



ENCODER RNN

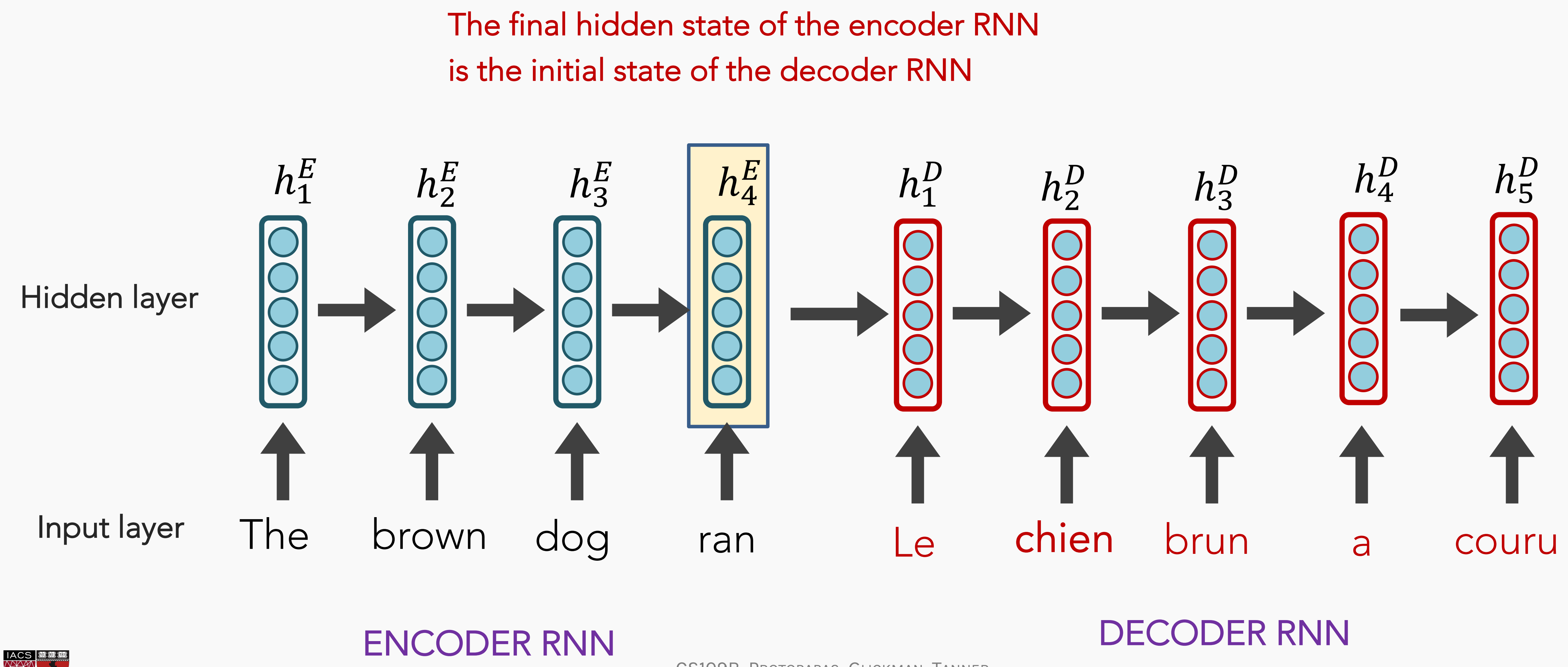
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN
is the initial state of the decoder RNN

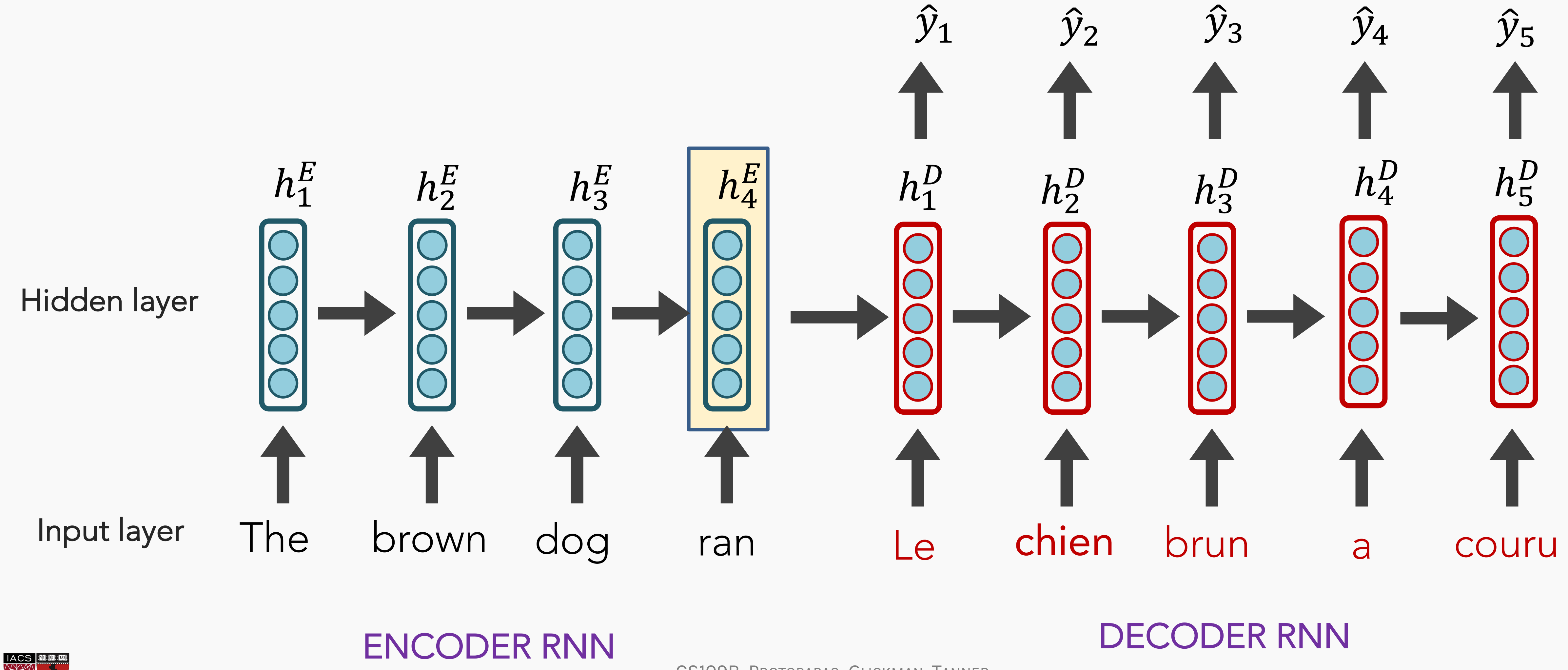


ENCODER RNN

Sequence-to-Sequence (seq2seq)

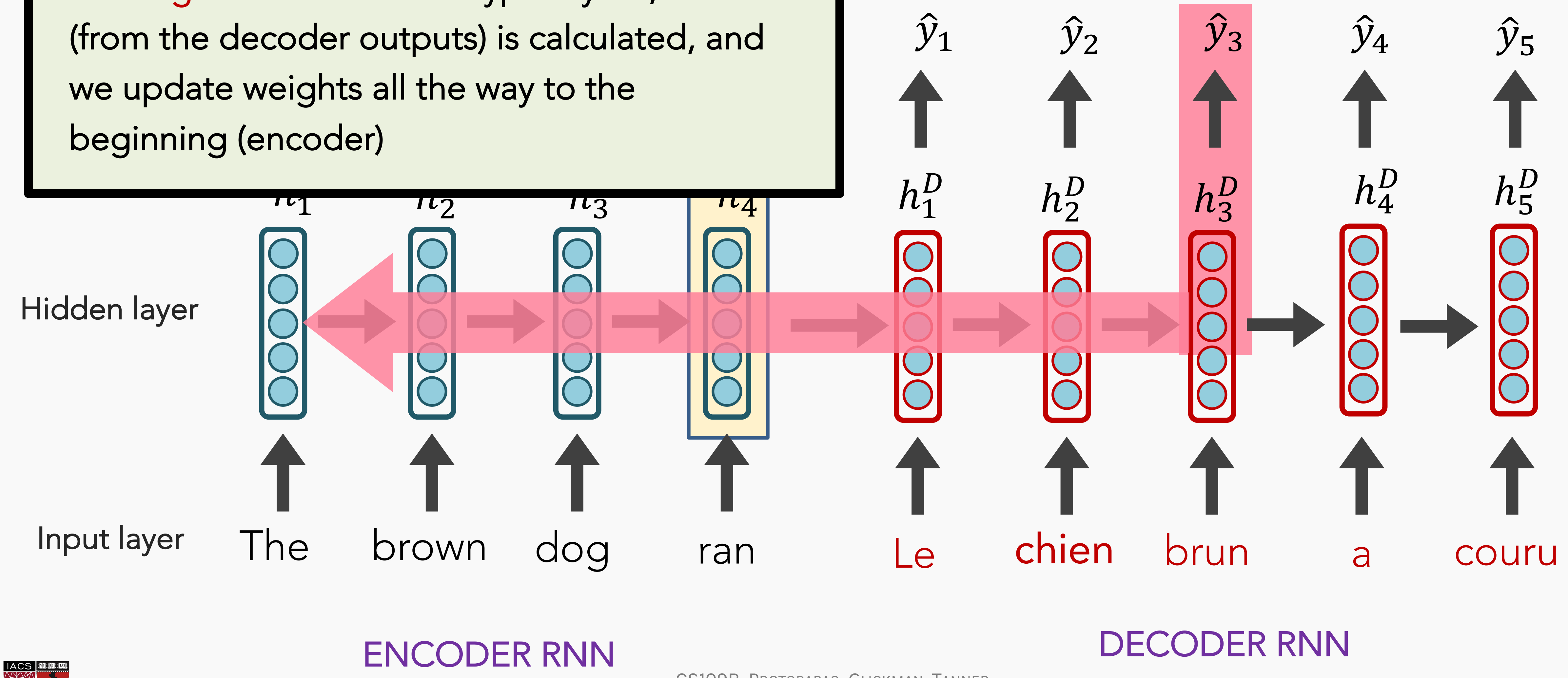


Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

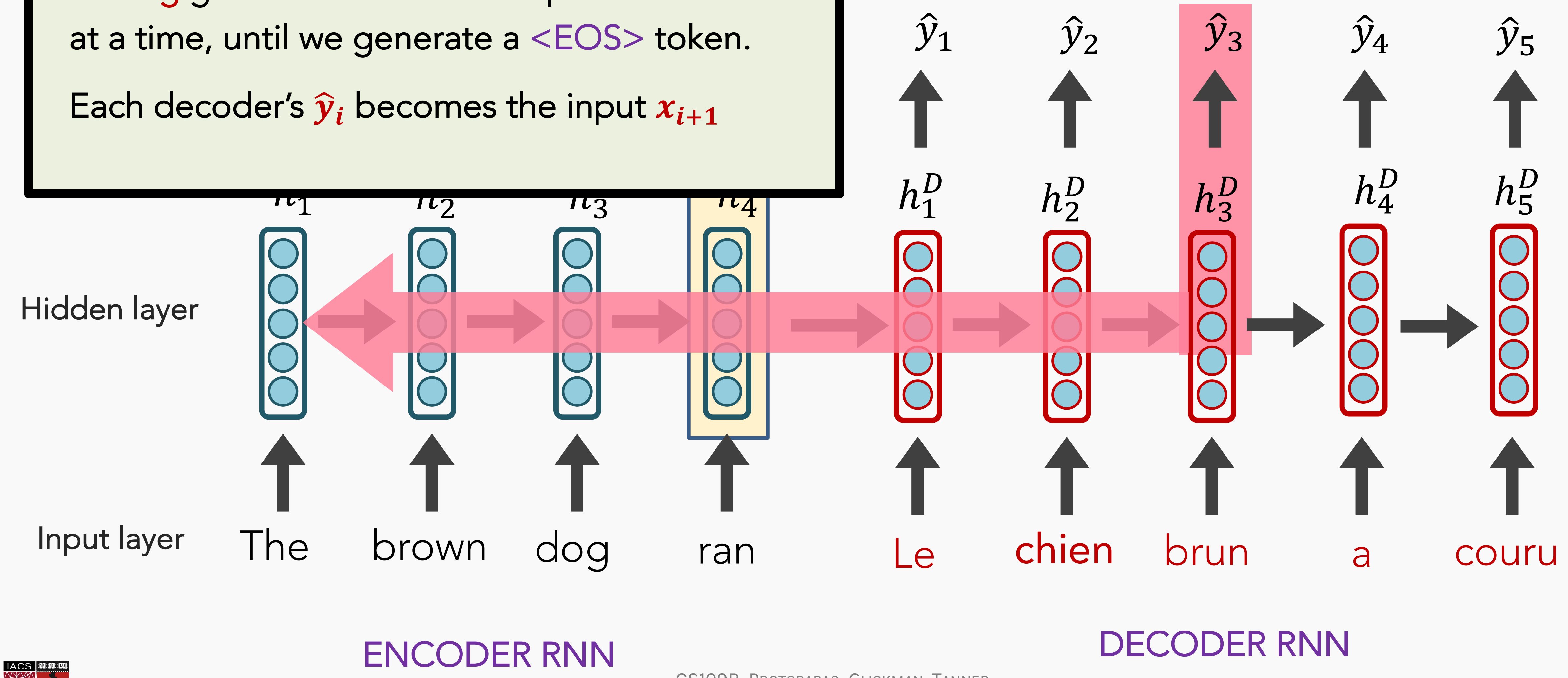
Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder)



Sequence-to-Sequence (seq2seq)

Testing generates decoder outputs one word at a time, until we generate a <EOS> token.

Each decoder's $\hat{y}_i$ becomes the input x_{i+1}

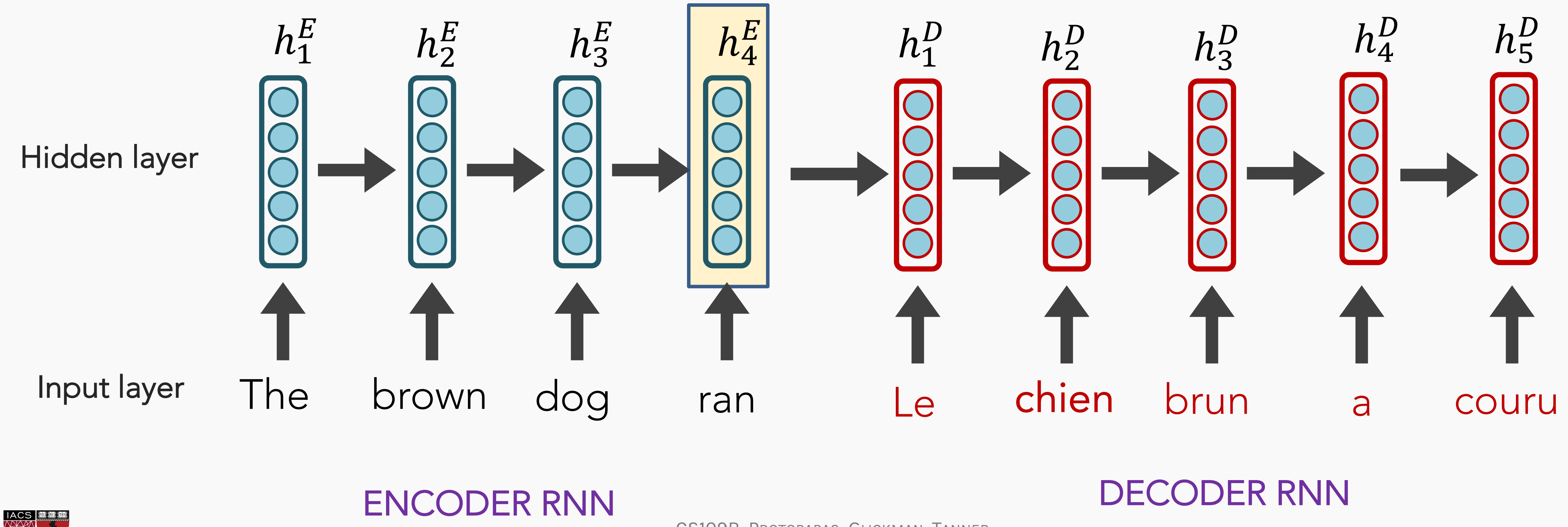


Sequence-to-Sequence (seq2seq)

See any issues with this traditional **seq2seq** paradigm?

Sequence-to-Sequence (seq2seq)

It's crazy that the entire "meaning" of the 1st sequence is expected to be packed into this one embedding, and that the encoder then never interacts w/ the decoder again. Hands free.

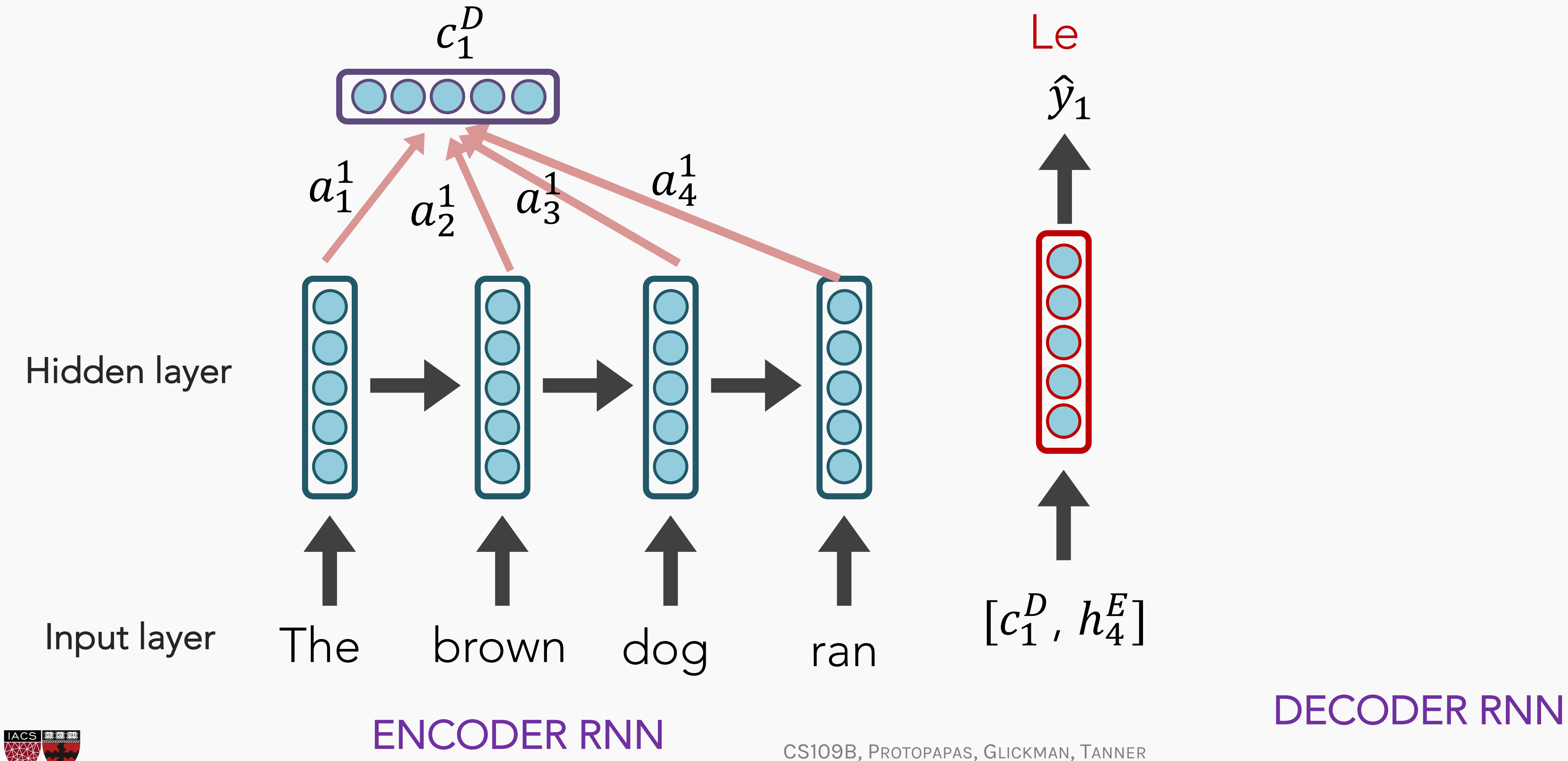


Sequence-to-Sequence (seq2seq)

Instead, what if the decoder, at each step, pays attention to a *distribution* of all of the encoder's hidden states?

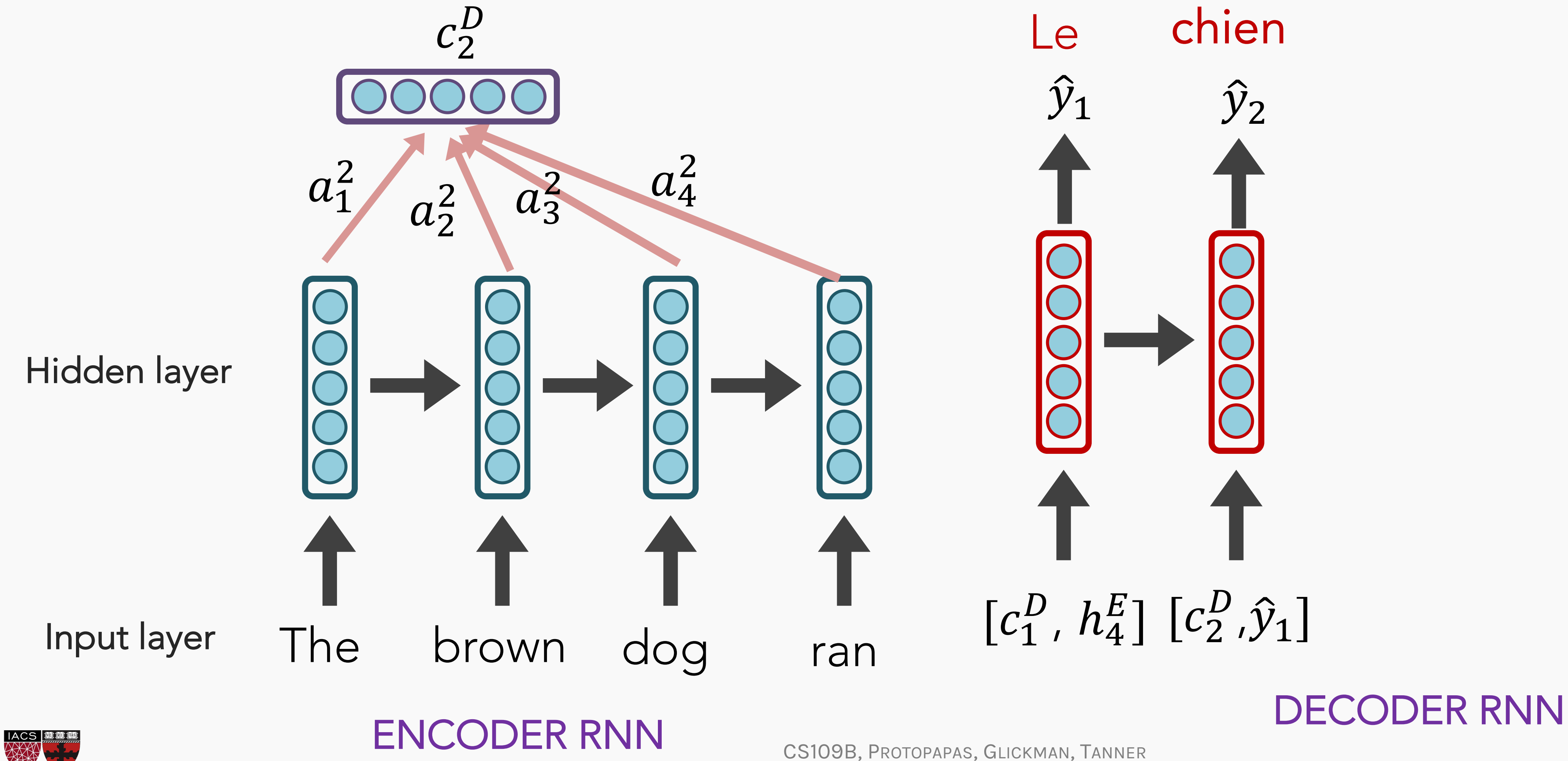
seq2seq + Attention

NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



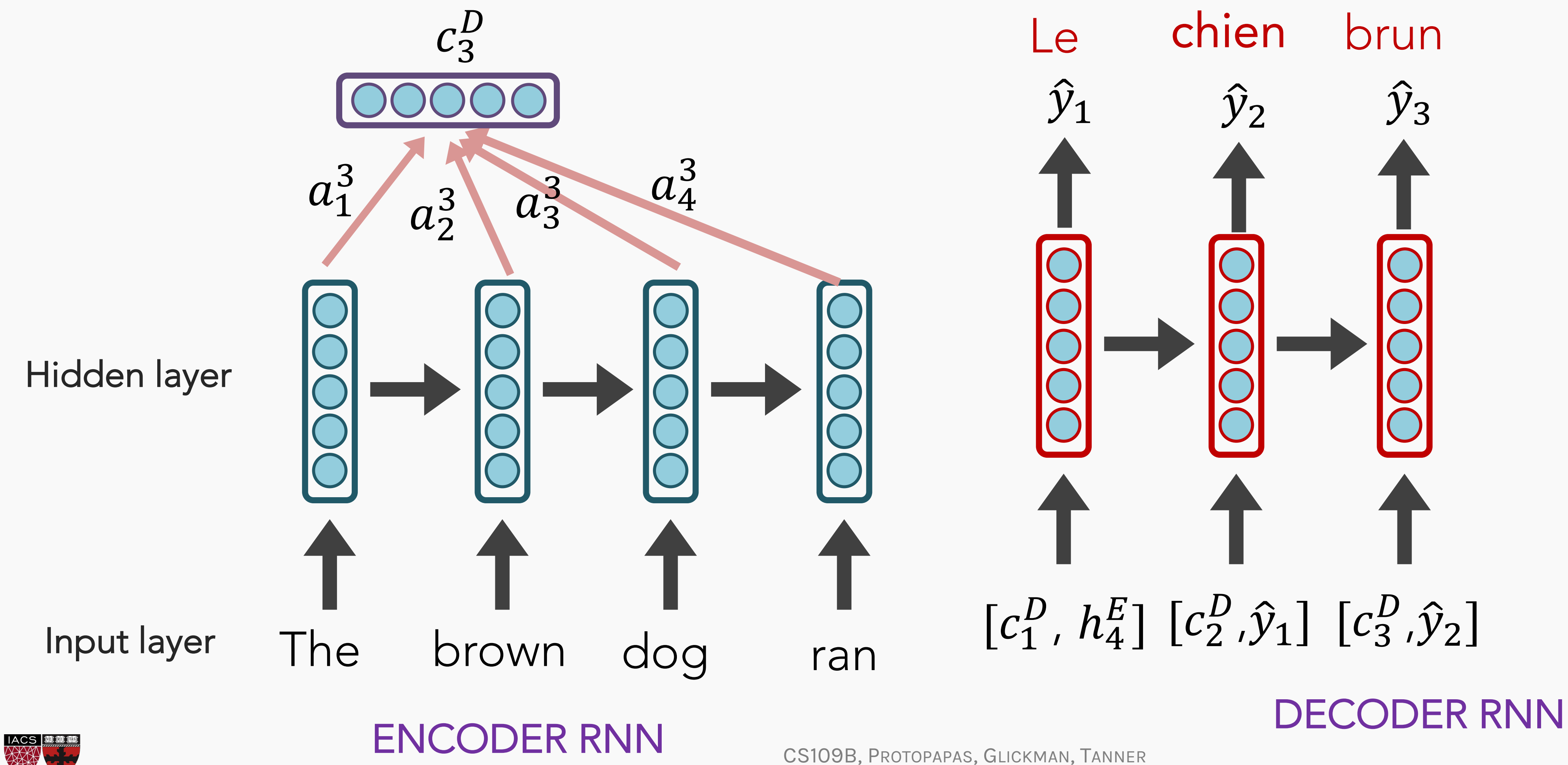
seq2seq + Attention

NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



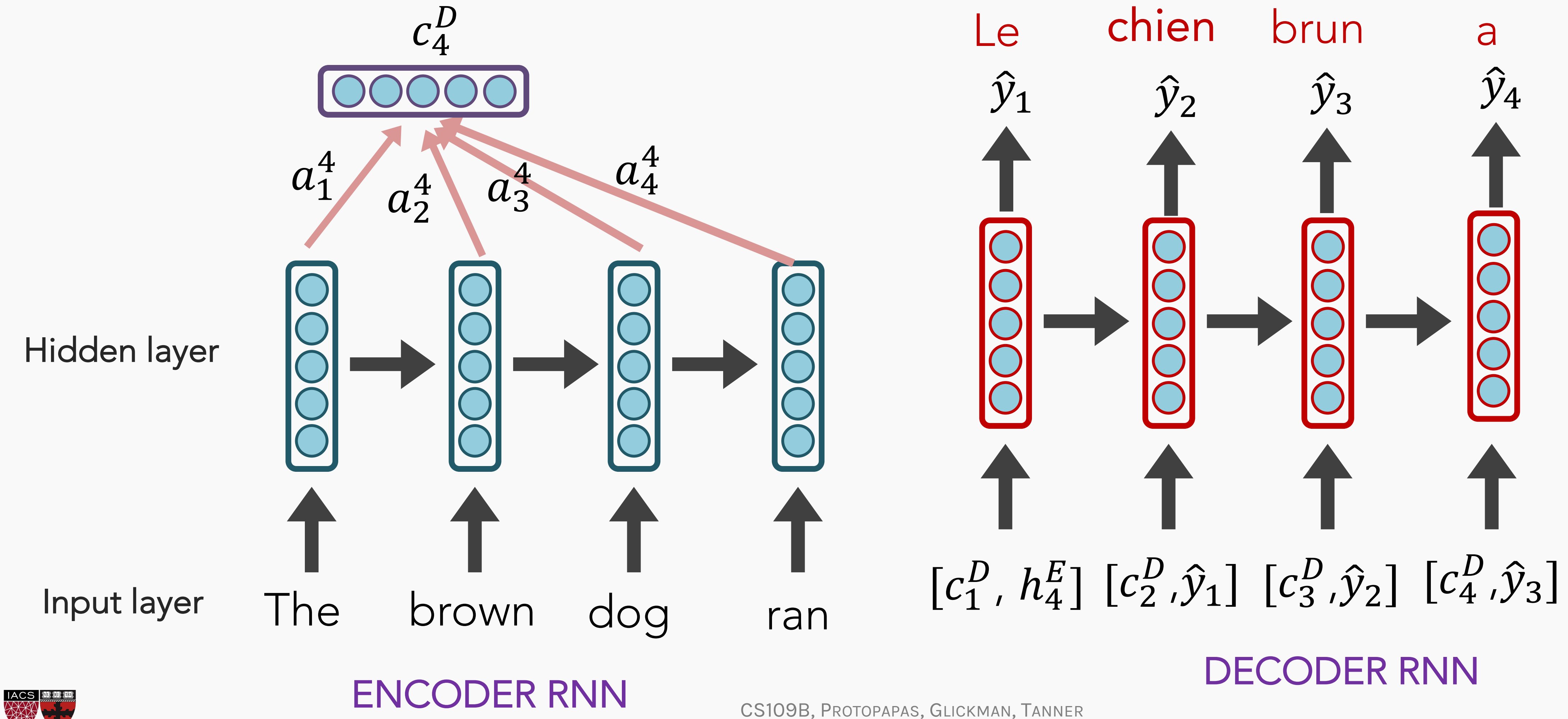
seq2seq + Attention

NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



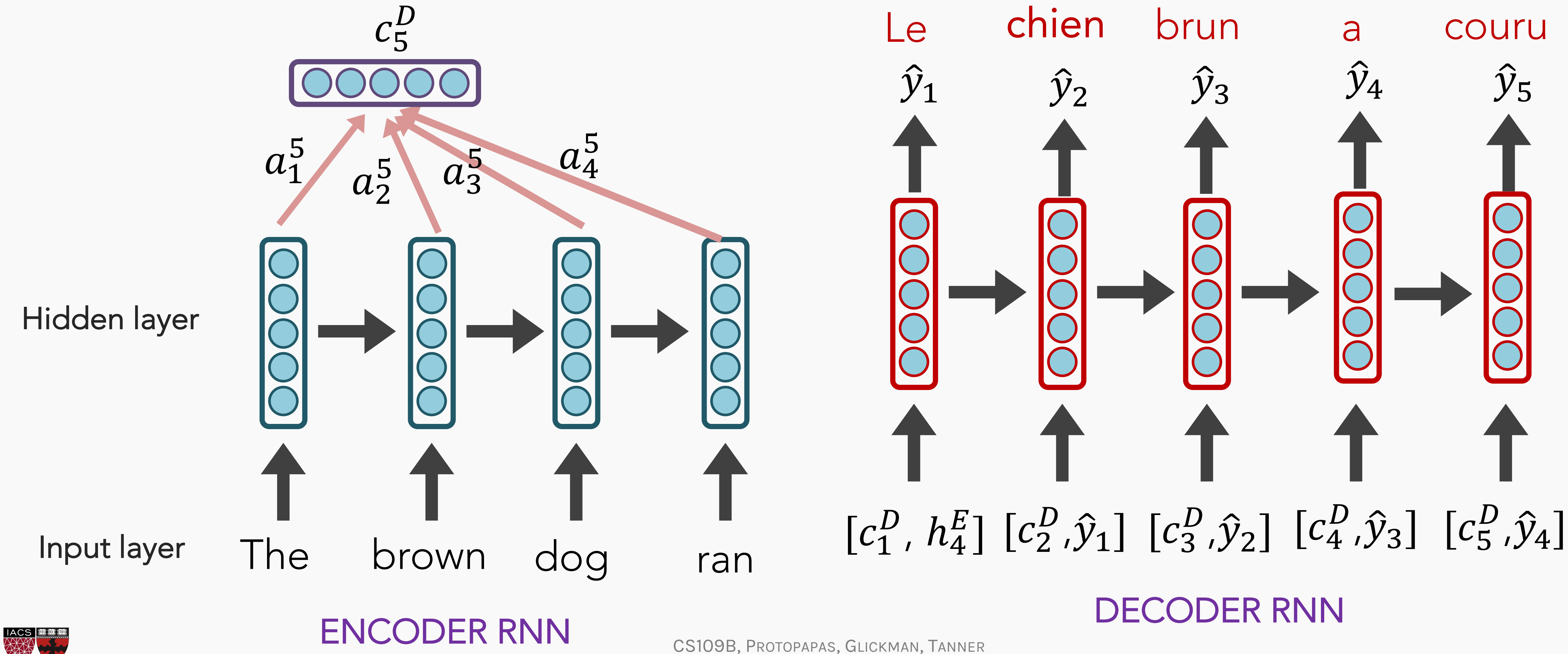
seq2seq + Attention

NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



seq2seq + Attention

NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



seq2seq + Attention

Attention:

- greatly improves seq2seq results
- allows us to visualize the contribution each word gave during each step of the decoder

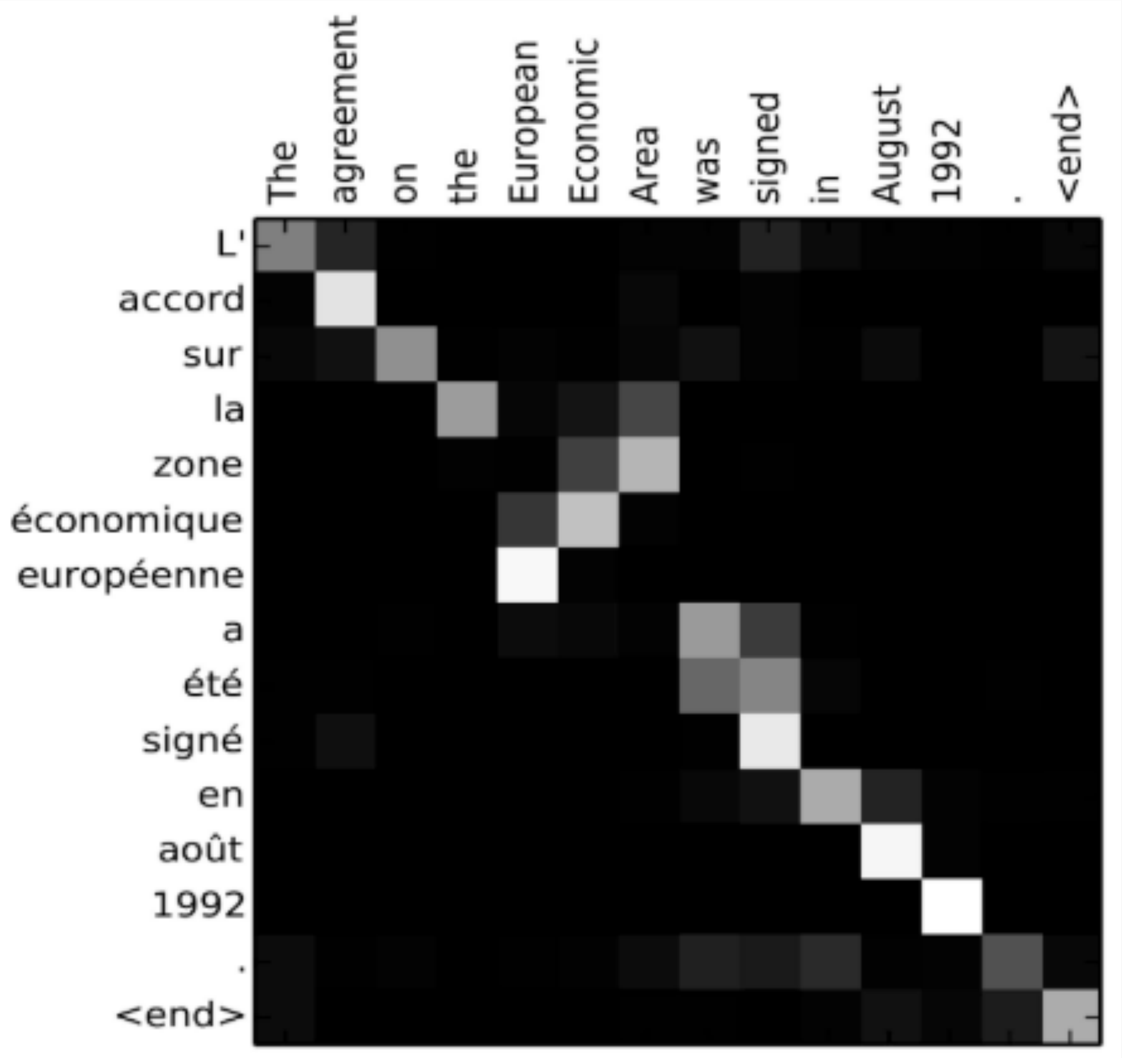


Image source: Fig 3 in [Bahdanau et al., 2015]()

Outline

Language Modelling

RNNs/LSTMs +ELMo

Seq2Seq +Attention

Transformers +BERT

Conclusions